



11-7-90
1740
P. 217₁

**Department of AERONAUTICS and ASTRONAUTICS
STANFORD UNIVERSITY**

SUDAAR 597

**EXPERIMENTS IN COOPERATIVE-ARM OBJECT
MANIPULATION WITH A TWO-ARMED
FREE-FLYING ROBOT**

Ross Koningstein

*Aerospace Robotics Laboratory
Department of Aeronautics and Astronautics
STANFORD UNIVERSITY
Stanford, CA 94305*

(NASA-CR-188025) EXPERIMENTS IN
COOPERATIVE-ARM OBJECT MANIPULATION WITH A
TWO-ARMED FREE-FLYING ROBOT Ph.D. Thesis
(Stanford Univ.) 217 p CSCL 131

N91-21529

Unclass
63/37 0001740

Research supported by
NASA Contract NCC 2-333

OCTOBER 1990

36

P-224


EXPERIMENTS IN COOPERATIVE-ARM OBJECT
MANIPULATION WITH A TWO-ARMED FREE-FLYING
ROBOT

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF AERONAUTICS AND ASTRONAUTICS
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

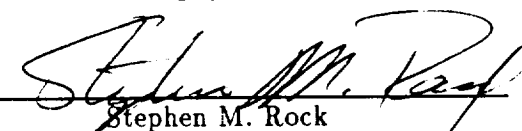
By
Ross Koningstein
October 1990

© Copyright by Ross Koningstein 1991
All Rights Reserved.


I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.


Robert H. Cannon, Jr.
Department of Aeronautics and Astronautics
(Principal Adviser)


I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.


Stephen M. Rock
Department of Aeronautics and Astronautics

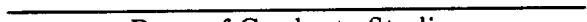
I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.


Emery I. Reeves
Department of Aeronautics and Astronautics

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.


Daniel B. DeBra
Department of Aeronautics and Astronautics

Approved for the University Committee on Graduate Studies:


Dean of Graduate Studies

2

Abstract

Developing computed-torque controllers for complex manipulator systems using current techniques and tools is difficult because they address the issues pertinent to simulation, as opposed to control. This dissertation presents a new formulation of computed-torque (CT) control that leads to an automated computed-torque robot controller program. This automated tool is used for simulations and experimental demonstrations of endpoint and object control from a free-flying robot.

A new computed-torque formulation states the multibody control problem in an elegant, homogeneous and practical form. A recursive dynamics algorithm is presented that *numerically* evaluates kinematics and dynamics terms for multibody systems given a topological description. Manipulators may be free-flying, and may have closed-chain constraints. With the exception of object squeeze-force control, the algorithm does not deal with actuator redundancy. The algorithm is used to implement an automated 2D computed-torque dynamics and control package that allows joint, endpoint, orientation, momentum and object squeeze-force control. This package obviates the need for hand-derivation of kinematics and dynamics, and is used for both simulation and experimental control in the course of this research.

Endpoint control experiments are performed on a laboratory robot that has two arms to manipulate payloads, and uses an air bearing to achieve very-low drag characteristics. The robot's base body mass and inertia are considerably larger than that of the manipulator arm segments, much like NASA's proposed Orbital Maneuvering Vehicle. Simulations and experimental data for endpoint and object controllers are presented for the experimental robot – a complex dynamic system.

There is a certain *rather wide* set of conditions under which CT endpoint controllers can neglect robot base accelerations (but not motions) and achieve comparable performance to including base accelerations in the model. The regime over which this simplification holds is explored by simulation and experiment. These simplifications can result in a savings of an *order of magnitude* of computation in the controller.

Momentum control via external forces and torques (e.g., thrusters) is provided for in the formulations, but is not done in this study.

2

To Joke and Arnold Koningstein

PAGE VI INTENTIONALLY BLANK

vii PRECEDING PAGE BLANK NOT FILMED

2

Civilization advances by extending the number of important operations which we can perform without thinking of them.

– *Alfred North Whitehead*

PAGE viii INTENTIONALLY BLANK

2

Acknowledgements

First, I wish to thank my principal advisor, Professor Robert H. Cannon, Jr., for his support and guidance during my time at Stanford. It is his insight that made the Aerospace Robotics Laboratory possible: this stimulating research laboratory has proven to me that the most effective way to make big steps forward is to take several smaller steps, learning along the way. I am further indebted to Professors Cannon, Rock, Reeves and Debra for their constructive criticisms, in-depth discussions, and review of this manuscript.

The design and construction of the experimental free-flying robot was a cooperative project with Marc Ullman, without whom it would not have been possible. The countless late nights of work that went into it resulted in a fully functional experimental robot system, and combined the best of the talents of all the participants. My thanks go to my fellow students Vincent Chen for his construction of the vision system hardware and its software and help documenting the electronic circuitry; Bill Dickson for his help in the design and calibration of the end-point gripper; Warren Jasper for characterizing the mass distribution and calibrating the sensors on the robot; and Stan Schneider for designing and implementing various portions of the software. In addition, I thank all the above students—as well as Robert Zanutta, Chris Uhlik, Brian Anderson, and Celia Oakley—for their stimulating discussions and assistance on various occasions.

I appreciate the workmanship of Godwin Zhang, Yosi Druker, and Joseph Schlesinger for the design and construction of the analog sensor and actuator electronics. I also thank Gad Shelef for the design and assembly of the manipulator arms, and Peter Davidson for its fabrication.

I greatly appreciate this support for this research by the National Aeronautics and Space Administration under contract NCC 2-333.

I would like to thank my friends Andy Carlson, Scott Seligman, Roger Crew, Rob Zanutta, Larry Augustin and John Ortiz for their zanyness and willingness to explore bad movies, and for their friendship. Thanks also to Miriam Rivera for being a good friend. And finally, I would like to thank my parents for their encouragement and understanding.

2

PAGE xii INTENTIONALLY BLANK

2

Contents

Abstract	v
Acknowledgements	xi
List of Tables	xix
List of Figures	xx
List of Symbols	xxiii
1 Introduction	1
1.1 Motivation	1
1.2 Background	2
1.3 Contributions	7
1.4 Reader's Guide	9
2 Modeling: Kinematics and Dynamics	13
2.1 Assumptions	13
2.2 Kinematics	14
2.3 Dynamics	17
2.4 Nonholonomic Motion Constraints	21
2.5 Summary	28
3 Computed Torque Control	29
3.1 Components of a Computed Torque Controller	30

3.2	The Jacobian Matrix	32
3.3	Jacobian Augmentation Equations	35
3.4	Resolving Generalized Accelerations	42
3.5	Examples of Augmented Jacobian Matrices	44
3.6	Inverse Dynamics	46
3.7	Summary	49
4	Recursive Dynamics	51
4.1	The Kinematic Chain	51
4.2	Computed-Torque Control on Kinematic Chains	52
4.3	Recursive Formulation of Kinematic Terms	53
4.4	Application to Computed-Torque Control	58
4.5	Recursive Dynamics (RD) in Two Dimensions	61
4.6	Implementation of an Automated Computed-Torque Control System	66
4.7	Summary	75
5	Experimental Hardware	77
5.1	System Requirements	77
5.2	Design Approach	78
5.3	Robot System Overview	78
5.4	Air-Bearing	80
5.5	Manipulator Subsystem	81
5.6	Sensor Subsystems	83
5.7	Analog Electronics Layer	85
5.8	Computer and Communications Architecture	88
5.9	Target Vehicles	89
5.10	Summary	90
6	Endpoint and Object Control Experiments	93
6.1	Independent Endpoint Control	94
6.2	Cooperative-Arm Object Manipulation	104
6.3	Summary	115

7	Can Base Accelerations be Neglected?	116
7.1	Simplified Computed-Torque Endpoint Control	117
7.2	Simplified Modeling	121
7.3	Errors due to Mismatching	123
7.4	Experimental Results	129
7.5	Extremes in Mass Distribution	135
7.6	Summary	141
8	Conclusions	144
8.1	Summary	144
8.2	Recommendations for Further Research	146
A	Power and Energy Expressions	147
A.1	Kinetic Energy	147
A.2	Power	147
A.3	Nonlinear Terms	148
B	Circuit Diagrams	151
B.1	Power Control Unit	151
B.2	Battery Charging and Monitoring	152
B.3	LEDs and Inertial Sensors	157
B.4	Safety Disconnect Board	157
C	RD Computer Software	163
C.1	C Language Interface Specification	164
C.2	Matlab Interface Specification	170
C.3	Evaluation of Kinematics	173
C.4	Evaluation of Dynamical Equations of Motion	182
D	Multibody Simulation under Matlab	191
D.1	Independent Endpoint Control Simulation	192
D.2	Cooperative-Arm Object Control Simulation	195

List of Tables

4.1	Number of Floating-Point Operations in RD computed-torque control . . .	74
5.1	Free-Flying Robot Mass and Geometry Parameters	91
6.1	Trajectory Specification for the Two Endpoints	95
6.2	Second-Order Error-Controller Gains	100
6.3	Trajectory Specification for the Manipulated Object	105
6.4	Error Controller Gains for Position and Orientation and Squeeze Force . . .	109
7.1	Nominal Mass Distribution in a Fictitious Robot	143

List of Figures

1.1	Two-Armed Free-Flying Robot Model	8
2.1	Partial Velocities and Generalized Speeds	15
2.2	A Closed Kinematic Chain	21
2.3	A Constraint Force	24
3.1	Block Diagram of Computed-Torque Control	30
3.2	Inverse and Forward Dynamics	47
4.1	A Standard Configuration Assembly Robot Arm	52
4.2	A Single Link in a Serial Chain	55
4.3	Inverse Dynamics on a Link-by-Link Basis	58
5.1	Satellite Robot Model	79
5.2	Granite Surface Plate	80
5.3	Pressurized Gas Layer	81
5.4	Air Bearing	82
5.5	Manipulator Arm	82
5.6	Global Vision Camera	84
5.7	Analog Electronics Layer	85
5.8	Power System Overview	87
5.9	Software Development Computer Network	88
5.10	Free-Flying Objects used in Manipulation Experiments	89
6.1	Overview of Computed-Torque Endpoint Controller	94
6.2	Endpoint Position Control from a Free-Flying Robot: Feedforward versus Feedback	99

6.3	Animated Matlab Simulation Run of Endpoint Control from a Free-Flying Robot	101
6.4	Two Arm Endpoint Position Control from a Free-Flying Robot	102
6.5	Disturbance Rejection: One Arm Endpoint Disturbed on the Free-Flying Robot	103
6.6	Overview of Computed-Torque Object Controller	104
6.7	Animated Matlab Simulation Run of Object Control from a Free-Flying Robot	110
6.8	Cooperative-Arm Object Position Control from a Free-Flying Robot	111
6.9	Cooperative-Arm Object Squeeze-Force Control from a Free-Flying Robot .	113
6.10	Cooperative-Arm Object Control from a Free-Flying Robot: External Disturbance Rejection	114
7.1	An Error-Based Feedback Controller	124
7.2	Endpoint Acceleration Deviations from the Norm when Neglecting Base Accelerations, as a function of Base Mass Parameters	126
7.3	Endpoint Acceleration Deviations from the Norm when Neglecting Base Accelerations, as a function of Payload Mass	128
7.4	Expected Endpoint Controller Performance when Neglecting Free-Flying Robot Base Accelerations	130
7.5	Two-Arm Tracking Controller Data where Base Accelerations are Neglected in the CT Controller	131
7.6	Differences in Endpoint Position Error between Including and Neglecting Free-Flying Robot Base Accelerations	132
7.7	Object Controller Data where Base Accelerations are Neglected in the CT Controller	133
7.8	Differences in Object Positioning Error between Including and Neglecting Free-Flying Robot Base Accelerations	134
7.9	Endpoint accelerations in Response to Commanded Accelerations as a Function of Base Mass.	136
7.10	Condition Number of Augmented Jacobian as a Function of Robot Base Mass.	137
7.11	Robot with Light Base Mass Attempting Endpoint Motion	138

7.12 Robot with Medium Base Mass Attempting Endpoint Motion	140
B.1 Power Control Unit: Switches and LEDs	153
B.2 Power Control Unit: External Power	154
B.3 Power Control Unit: Battery Control (two total)	155
B.4 Power Control Unit: Connector Pinout	156
B.5 Battery Charger: Charge and Sensing Circuitry (two total)	158
B.6 Interface Circuits for LEDs and Inertial Sensors	159
B.7 Safety-Cutout Circuit: Disables Motors and Thrusters	161

List of Symbols

This list contains the symbols used in this thesis. The meaning of the term is given, and the section in which it is defined is indicated, if appropriate.

$\hat{x}, \hat{y}, \hat{z}$	Inertial unit vectors, §2.2.1
$\hat{\lambda}_i$	Axis of rotation i , §2.2.1
\mathbf{r}^i	Inertial position of point i , §2.2.1
\mathbf{v}^i	Velocity of point i , §2.2.1
\mathbf{a}^i	Acceleration of point i , §2.2.2
ω^j	Angular velocity of body j , §2.2.1
α^j	Angular acceleration of body j , §2.2.2
q_r	Generalized coordinate r , §2.2.1
u_r	Generalized speed r , §2.2.2
\dot{u}_r	Generalized acceleration r , §2.2.2
\mathbf{v}_r^i	Partial velocity r of point i , §2.2.1
\mathbf{c}^i	Constraint velocity of point i , §2.2.1
\mathbf{c}_r^i	Partial constraint velocity r of point i , §2.2.1
ω_r^j	Partial angular velocity r of body j , §2.2.1
\mathbf{F}^i	Force on point i , §2.2.1
\mathbf{T}^j	Moment on body j , §2.2.1
\mathbf{F}_r	Generalized active force r , §2.2.1
\mathbf{F}_r^*	Generalized inertia force r , §2.2.1
\mathbf{L}_r^S	Partial linear momentum r of system S , §2.2.1

\mathbf{H}_r^{S/S^*}	Partial angular momentum \mathbf{r} of system S about point S^* , §2.2.1
\mathbf{I}_j/j^*	Inertia of body j about its center of mass j^* , §2.2.1
\mathbf{W}	Mapping of generalized speeds to generalized coordinates, §2.2.1
\mathbf{J}	Jacobian matrix, §2.2.1
\mathcal{J}	Augmented Jacobian matrix, §2.2.1
K	Kinetic energy, §2.2.1
P	Power, §2.2.1

List of Acronyms

This list contains the acronyms used in this thesis. The section in which the acronym is defined or first used is also indicated.

A/D	Analog-to-Digital converter, §5
ARL	Aerospace Robotics Laboratory, §1
CCD	Charge-Coupled Device, §5
CPU	Central Processing Unit, §5
D/A	Digital-to-Analog converter, §5
FTS	NASA's Proposed Flight Telerobotic Servicer, §4.1
Hz	Hertz or cycles per second,
LED	Light Emitting Diode, §5
NASA	National Aeronautics and Space Administration, Acknowledgements
OMV	NASA's Proposed Orbital Maneuvering Vehicle, §4
PD	Proportional-plus-Derivative control, §6
RVDT	Rotary Variable-Differential Transformer, §5
SCARA	Standard Configuration Assembly Robot Arm, §4
UNIX	A popular multiprocessing operating system [14] ¹ , §5
VME	A popular 32-bit backplane bus architecture, §5
WIND	Wind River System's operating system kernel ² , §5

¹ *UNIX* is a trademark of AT&T, Inc.

² *WIND* is a trademark of Wind River Systems, Inc.

2

Chapter 1

Introduction

1.1 Motivation

The term Robot, coined by Carl Capek¹, refers to a machine invented to perform repetitive work. Robots in various incarnations have since been used in industry to replace human workers in highly repetitive or very dangerous operations, or to extend manufacturing technologies into regimes beyond human capability.

Advances in robotic manipulation technology continue to extend the capabilities of manipulator systems, both on earth, and in space. An example of a robotic system that requires advances in control techniques is NASA's planned Orbital Maneuvering Vehicle (OMV), a two-armed free-flying robot. It can use its cooperating arms to manipulate fragile objects, such as spacecraft, with lower applied stresses than if using a single arm. Multiple arms do, however, increase the complexity of the dynamic system that needs to be taken into account when designing a manipulator control system. The additional dynamics introduced by the free-flying nature of the robot further complicate the situation.

Manual derivation by an analyst of such dynamic system equations for control is a very time-consuming process (and susceptible to error). This thesis develops a unifying multibody computed-torque controller formulation, and presents a computer program that uses this formulation to calculate numerical equations for computed-torque control automatically. This program can act both as a design tool for simulation, and as a real-time

¹from the Polish verb *robot*: to work

controller, obviating the need for derivation by hand. Thus, simulations and control of complex dynamic systems such as free-flying robots with cooperating manipulator arms can be done with relative ease.

1.2 Background

1.2.1 Literature Review

Manipulation from a Free-Flying Robot

Historically, investigation into the control of manipulation from a free-flying robot has been via the computed-torque method. This method is known to compensate well for the nonlinearities and time-varying dynamics found in manipulator systems, and has been verified experimentally by Khatib [15], Craig [6], Khosla and Kanade [16] and others. It has also been used successfully to control real-world experimental systems by Schneider [29], Uhlik [33] and other students in the ARL.

Extending the computed-torque control scheme to manipulation from free-flying robots has not occurred without problems, however. The problem of controlling a manipulator mounted on a free-flying robot exposes a problem in forming and solving a Jacobian matrix equation: the manipulator Jacobian matrix becomes non-square with a free-flying robot because the endpoint degrees of freedom combined do not account for all the degrees of freedom in the system. The so-called ‘redundant’ degrees of freedom mentioned in the literature are attributable to the free-flying base, and most of the approaches have come up with methods to solve for resolved rates or accelerations by removing the extra degrees of freedom introduced by the base from the formulation. Resolved rates or accelerations are computed using the system’s Jacobian and its derivative as discussed in Craig and Khatib. The manipulator Jacobian, \mathbf{J} is defined by the equation

$$\mathbf{v}^{\text{endpoints}} = \mathbf{J}\dot{\mathbf{q}}$$

where \mathbf{v} is a vector of the speeds of the manipulator endpoints, measured in some coordinate system and $\dot{\mathbf{q}}$ are the derivatives of the manipulator joint rates and the free-flying robot’s base coordinate rates. In fact, the Jacobian matrix is a generalized derivative, and

relates many functions' derivatives to a set of independent variables. In the case of robotic manipulators, the derivatives have been endpoint velocities or angular velocities, and the independent variables have been the joint rates.

Approaches to dealing with a non-square Jacobian matrix are numerous. Alexander [1] partitioned the mass matrix and isolated needed additional constraints to compute control torques. No assumptions were made about momentum, as long as the controller was informed of changes. Umetani and Yoshida [36] modified the definition of the Jacobian into a Generalized Jacobian to eliminate the redundant momentum states; subsequent experimental work [37] illustrated the validity of such an approach. Their resolved-rate implementation assumed that the angular momentum of the total system was zero.

Masutani et al [41] used the generalized Jacobian to implement a Jacobian-transpose style controller, and presented simulation results. Woerkom and Guelman [38] used a generalized inverse to solve the Jacobian equation, with the result that their resolved accelerations were not consistent with the system dynamics, because they chose to have resolved accelerations solved using minimization relations that were not consistent with the system dynamics. Their conclusions from simulation results indicated that this was not a desirable method.

Carignan [3] used a sliding-mode controller that only partially compensated for manipulator-body interactions; his experimental results unfortunately demonstrated limitations in the experimental hardware. He also presented a theoretical formulation for closed-chain kinematics using a Lagrangian formulation but did not explore this avenue further.

Vafa and Dubowski [7] defined a Virtual Manipulator as an abstract model of a free-flying robot with manipulator. This model assumed that external forces and torques were zero, and that the linear momentum of the system was zero. A simple controller based on this model was developed in theory; however, it assumed that the robot base is separately controlled to counteract angular motion.

Koningstein and Ullman [17] presented a method for augmenting the manipulator Jacobian, creating a System Jacobian, in order to solve the control problem for both free-flying and closed-chain manipulator systems.

None of the methods cited, with the exception of that by the author, have allowed

momentum to be controlled in the manipulator computed-torque formulation.

One further point: although it has been established that the robot base motions are affected by manipulator motion, and is also subject to drift due to initial conditions, there have been no experiments evaluating the relative performance of controllers that include full free-flying dynamics model versus those that have simplifications. This poses the interesting question: which aspects of the extra dynamic modeling for free-flying manipulators are necessary if computation costs are important ?

Cooperative Manipulation

Cooperative manipulation is the act of controlling an object such that all arms work together, and that the controller resolves the dynamics of the closed-chain constraint. This has the powerful ramification that the task specification to the controller is in the form of desired object behavior, and does not concern itself with the activity of the manipulators.

Luh and Zheng [19] formulate the closed-chain constraint for dynamics by differentiating a position constraint at a cut in the closed chain. This method is also used by Tarn, Yun and Bejczy [32] to develop a method that allows force control independent of task space. Hayati [8] explicitly computed the torques and forces required to move an object and divided them up among the used manipulators, a method also used by Schneider [29], who performed a series of experiments demonstrating the feasibility of such a scheme.

Carignan and Akin [4] also present a strategy for cooperative control of two arms, but to carry a load in zero g. The method of Hayati was also derived, albeit in a different form, by Seraji [30] in order to control force in the constraint directions (eg. the object squeeze direction) and desired position in the other degrees of freedom. Nakamura et al [40] derived terms for feedforward control, but not feedback, in an early derivation. Later, Nakamura and Ghodoussi [20] derived the constrained dynamics equations of motion using a Lagrangian approach.

Typically, descriptions of dynamic systems with constraints are reduced in order (i.e. rank) for solution. This is because the constrained system has a reduced number of degrees of freedom, and unique consistent solutions for the joint accelerations require that the constraints hold. Nielan [21] discusses the cost of formulating constrained equations of

motion. It involves formulating the unconstrained mass matrix and nonlinear terms, and the modifying them using matrix operations: an expensive process.

In fact, constrained equations are not necessary in order to *control* constrained systems using computed-torque controllers. Hayati and Schneider demonstrated this for simple dynamic systems. This thesis presents a general method for including the dynamic constraints in the Jacobian equation, as opposed to the dynamics equation, so that solving the Jacobian equation will always result in a consistent set of resolved accelerations. This conclusively illustrates that it is not necessary to formulate the constrained equations of motion, regardless of how many dynamic constraints there are in the system, in order to apply computed-torque control.

Recursive Dynamics Formulations

To study the dynamics and control problem of a physical system, accurate dynamic modeling is required. Manipulator dynamics offer interesting challenges because of their nonlinearity and coupled nature. Hollerbach [10] formulated manipulator dynamics recursively using the Lagrange method. Wampler [39] formulated kinematics and dynamics terms for manipulators using partial velocities and provided computational cost measures for these operations. These recursive derivations set the stage for work in automating equation formulations, since it is easiest to implement computer programs that operate recursively. Recursive algorithms treat each body equally, while noting their relationships to previous and successive bodies. Thus, computer programs using this approach have but to deal with one case: a body in a chain of bodies, where conditions depend on the previous body in the chain.

Computer codes for automatic generation of symbolic equations of motion have appeared, all of them based on Kane's generalized dynamics formulation [13]. Rosenthal's derivation [27] evolved into SDEXACT, while a similar program called SYMBA was developed by Nielan [21]. These codes, however, do not address all the needs of the control system designer since they approach the problem from a simulation viewpoint. They consider the formulation of the dynamics equation, which is useful for simulation and for the inverse dynamics aspect of computed-torque control, but which does not deal with

generating resolved accelerations.

Currently, a designer must derive the Jacobian entries by hand, and can then use symbolic manipulation packages (eg. MACSYMA) to factor and compact the solutions for efficient run-time. Using a new tool, AutoLev [28], it is possible to formulate partial velocities semi-automatically and interactively. Partial velocities have been shown by Wampler [39] to be useful as entries in a Jacobian equation, while Kane, Nielan and Rosenthal have shown them to be useful for dynamics formulation.

The inverse dynamics solution does not necessarily involve formulating the mass (inertia) matrix and evaluating the nonlinear terms of the dynamics equations. A significant amount of computation can be saved by not generating or solving this equation, as has been demonstrated by Luh, Walker and Paul [18] in their numerical implementation of recursive Newton-Euler inverse dynamics.

Numeric solutions differ from the symbolic solutions previously discussed as follows: symbolic programs generate specific computer codes for specific problems, whereas the recursive numeric methods provide the designer with an algorithm (a fixed program) which solves the general case.

This approach is the one taken in this thesis, where an algorithm is used to generate a numerical Jacobian equation, for acceleration resolution, and a recursive Newton-Euler algorithm to solve for joint torques. Computed-torque control is presented as an algorithmic process based on recursion using partial velocities.

1.2.2 Key Issues

- There is no single body of theory that neatly describes how to formulate the computed-torque control problem for a free-flying, cooperating-arm robot.
- Automation tools exist for dynamics simulation but these tools do not address the formulation of the Jacobian, nor an efficient mechanism for implementing inverse dynamics. Both are critical to computed-torque control. Control system designers are therefore faced with complex mathematical derivations in order to formulate them.
- While automated equation generators exist commercially, none specifically addresses

the additional work required for computed-torque control, above and beyond that required for numerical simulation. There is no reason that a Jacobian cannot be automatically formulated and solved along with the inverse dynamics.

- There has been little experimental work in evaluating the relative performance of different types of endpoint feedback controllers on free-flying robots, partially due to the difficulty in formulating controllers for such systems.

1.3 Contributions

The research reported here makes the following original contributions to the fields of automatic control and robotics:

1. A novel method is developed for dealing with dynamic constraints, such as closed kinematic chains, in a computed-torque controller by augmenting the Jacobian matrix. As a consequence, significantly simpler unconstrained dynamics can be used to solve for joint torques.

2. The addition of linear and angular momentum to the set of quantities that can be controlled via a computed-torque controller is made possible by augmenting the Jacobian matrix with *partial momenta* terms. Previous formulations for free-flying robot controllers have expected external momentum control, or put restrictions on momentum.

3. A new computed-torque formulation is presented that *unifies* fundamental components of the kinematics, Jacobian, inverse dynamics and forward dynamics in a recursive algorithm using partial velocities. The formulation of the Jacobian matrix equation requires very little computation with this method.

4. A computer program for two-dimensional recursive dynamics (RD) has been developed that automatically solves numerically both a Jacobian equation and the inverse dynamics, given a description of the system topology, the mass distribution, and the desired control space (i.e., joint, endpoint, momentum). This program is useful for both simulation and real-time control. It obviates the need for manual derivation and coding of the kinematics, dynamics, Jacobian, and quantities of interest. This algorithm is applicable to fixed-base and free-flying robots, and also to closed-chain manipulator systems.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

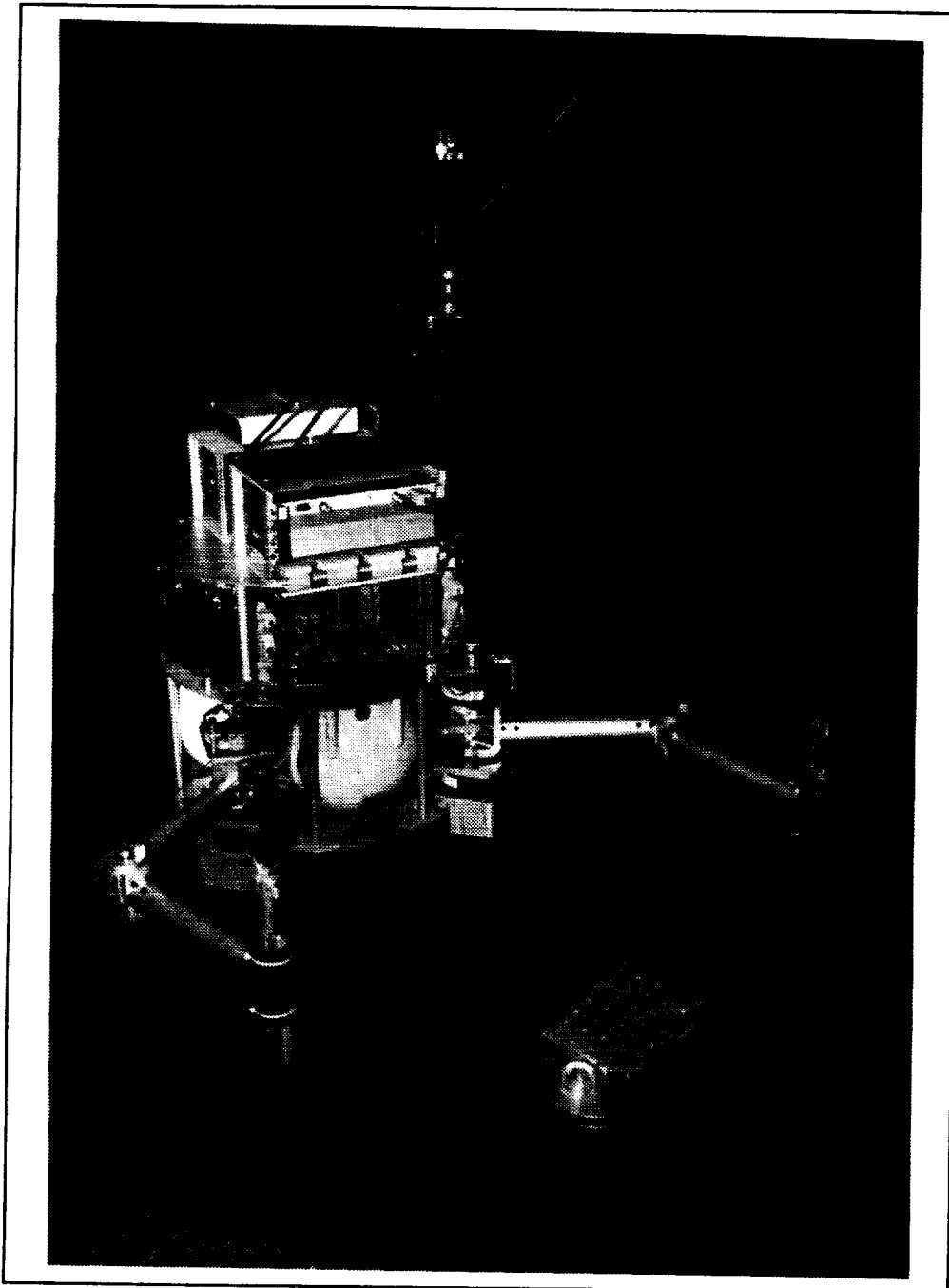


Figure 1.1: **Two-Armed Free-Flying Robot Model**

This experimental autonomous robot, which uses an air bearing for faithful simulation of zero-gravity in two dimensions, is used to verify the performance of controllers developed in this thesis.

5. An experimental free-flying robot with two manipulator arms has been designed and constructed² to act as a testbed for various control systems. This experimental robot, depicted in Figure 1.1, features an air-cushion for near frictionless 2D behavior, as well as two low-friction, direct-driven manipulator arms with force sensing grippers. The free-flying robot model is a completely autonomous system, carrying its own gas for flotation, batteries for electrical energy, and electronic and computer systems for control.

6. Experiments have demonstrated independent manipulator endpoint position control and cooperative manipulation of an object from a free-flying robot. Full free-flying multi-body dynamic models were used. When cooperatively manipulating an object, a point on the object is position controlled, the object's orientation is controlled, and the squeeze force exerted on the object is controlled. The computed-torque controller offers decoupled, linearized control of these otherwise coupled quantities.

7. The effects of simplifying the controller – partially neglecting the free-flying dynamics – are investigated. In particular the simplification of neglecting base accelerations is examined. When the manipulator arms are articulated in such a way as to isolate them from base angular motions, and the base body has significantly more mass and inertia than the manipulator arms, this approximation turns out to be quite reasonable. The simplification can result in negligible degradation of the endpoint controller's performance, while reducing the computational burden significantly. Simulation predictions of the effect of this simplification are made for several robot configurations. Specific examples in simulation, and two in experiment, serve to illustrate the effects. Experimental data confirm the validity of this simplification for specific examples of free-flying robot manipulator and payload under endpoint control.

1.4 Reader's Guide

1.4.1 Outline

This thesis is intended to serve two purposes. The first is to present a new unified computed-torque formulation and an automated computed-torque controller program as useful tools

²This was a cooperative project with Marc Ullman, also a Ph.D. candidate in the department of Aeronautics and Astronautics.

for continuing research into the control of complex multibody systems. The second is to provide experimental results demonstrating independent-arm and cooperative-arm manipulation from a free-flying robot. The more general material, a rigid-body dynamics and computed-torque control derivation, is presented first, and then specialized to the system under study: a two-armed free-flying robot. The second part describes the experimental setup and presents manipulator endpoint control and cooperative-arm object manipulation experimental results.

The first chapter has presented the background and motivation behind this research. Existing work involving free-flying manipulator control, cooperative manipulation, and recursive dynamics formulations are reviewed, and work developed in this thesis has been introduced. The contributions to knowledge brought about due to this research are described, and this reader's guide have been presented.

In chapters 2, 3 and 4, a new underlying formulation developed for computed-torque (CT) control of rigid-body systems is presented. It applies to fixed-base robots, free-flying robots, and robots using cooperating arms for manipulation. Chapter 2 contains kinematics and dynamics modeling and dynamical equations of motion for simulation. Chapter 3 contains the formulation of the computed-torque controller based on the kinematics modeling. The augmented Jacobian Matrix, useful for dealing with free-flying and/or constrained dynamic systems, is introduced at this point. The control algorithm is similar in form, although not in implementation, to standard computed-torque controllers, and encompasses into one homogeneous, simple formulation most of the work to date on free-flying and closed-chain systems. In chapter 4 recursive dynamics relations for kinematic chain manipulators are presented. A recursive algorithm for rigid-body kinematics and dynamics is implemented for two-dimensional robots. It is used for both computed-torque control (in simulation and experiment) and for dynamics simulation of multibody systems. This material is of interest to those studying the structure of dynamics and CT controllers for manipulators.

In chapter 5 the experimental apparatus, a free-flying robot model, is described. This free-flying space robot model is used to validate experimentally the CT controller developed in chapters 2, 3 and 4. The characteristics of the free-flying robot are discussed, and are

of interest to those interested in the experimental results, since the results are dependent upon the physical system. Experimental results are discussed in chapters 6 and 7.

In chapter 6 two types of experiments demonstrate control of a two-armed free-flying robot. In the independent arm endpoint position control experiments, the two arm endpoints are made to follow independent trajectories. In the cooperative manipulation experiment, an object is position and orientation controlled. Squeeze force is also controlled. The controllers are implemented by the automated recursive dynamics program developed in chapter 4. The system description files that configured this program to perform these complex control functions are presented.

In Chapter 7 the effects of simplifying the controller on the performance of endpoint control of free-flying robots is examined. Free-flying dynamic modeling for control is shown to differ from fixed-base modeling in two aspects: there are accelerations of the base, and there are extra system states associated with the robot base. Base angular velocity is shown to compensate for nonlinear terms, and is shown to be computationally inexpensive: it is included in the model. The effects of neglecting base accelerations, a simplified formulation, are studied. Simulation results show the effects of this simplification for a variety of base mass (and inertia) values, and a variety of payload masses. Conditions where base accelerations are important are discussed. Experimental demonstrations similar to those of chapter 6, but using simplified controllers, are presented.

In Chapter 8 conclusions are drawn from the experimental results and the predictions of the performance of free-flying robots using simplified controllers that neglect robot base dynamics. These conclusions are of interest to control designers and persons developing or designing free-flying robot control systems. Recommendations for future research are presented for those interested in continuing development of control systems for free-flying robots.

1.4.2 Notation

The notation employed throughout this thesis is intended to be as consistent as possible with that developed by Kane [13]. New terms have been introduced to facilitate the discussion and unify the style of presentation. Their definition and use is also intended to

be as consistent as possible with Kane's notational conventions. All velocities, momenta, and accelerations are expressed with respect to a Newtonian (inertial) reference frame.

The basic components of Kane's notation are the generalized coordinates for expressing position and orientation, and the generalized speeds for expressing motion. Velocities and accelerations denote the point they refer to. Angular velocities and accelerations denote the body they refer to. The partial velocities are the components of the velocities that can be attributed to each generalized speed. Generalized active and inertia forces are used to express Kane's dynamical equations of motion. Vectors are denoted by bold-face letters, and scalars by regular face letters. Vectors are 2D or 3D, according to the dimension of the system under analysis.

q_r	Generalized coordinate r	units of m or rad
u_r	Generalized speed r	units of $\frac{m}{s}$ or s^{-1}
\mathbf{v}^i	Velocity of point i	units of $\frac{m}{s}$
\mathbf{a}^i	Acceleration of point i	units of $\frac{m}{s^2}$
\mathbf{v}_r^i	Partial velocity r of point i ,	effectively $\frac{\partial}{\partial u_r} \mathbf{v}^i$
ω^j	Angular velocity of body j	units of $\frac{1}{s}$
α^j	Angular acceleration of body j	units of $\frac{1}{s^2}$
ω_r^j	Partial angular velocity r of body j ,	effectively $\frac{\partial}{\partial u_r} \omega^j$
$F_r + F_r^* = 0$	Kane's Dynamical equations of motion $r = 1..n$	
F_r	Generalized Active Force r , corresponding to generalized speed r	
F_r^*	Generalized Inertia Force r , corresponding to generalized speed r – units of $\frac{W}{m/s}$ or $\frac{W}{1/s}$ depending on choice of generalized speeds	

Chapter 2

Modeling: Kinematics and Dynamics

In this chapter kinematics and dynamics relations applicable to both simulation and control systems of multiple rigid bodies are presented. These relations differ from previous formulations in the way constrained (i.e. closed-chain) dynamic systems are modeled. The relations are applicable to free-flying or fixed robots, and can model a wide variety of dynamic constraints. New equations of motion suitable for simulating closed-chain systems are presented: these equations automatically ensure state consistency by using relaxation methods.

First, the assumptions under which the modeling is done are stated. This is followed by a derivation of general multibody kinematics and dynamics expressions, and a treatment of nonholonomic motion constraints. Expressions for kinetic energy and power input are presented in appendix A. The terms and equations that fall out of this analysis have useful structural properties which are used in chapter 4 to automate their formulation.

2.1 Assumptions

This modeling of kinematics and dynamics for simulation and control of multibody systems is based on the following assumptions:

1. Assume the bodies in the system can be treated as rigid.

2. Assume the controller, which is running at discrete time intervals, is running fast enough to function as if it were continuous in time.
3. Assume that friction within the dynamic system is negligible.
4. Assume that joint actuators can deliver perfect torques.

The mathematical modeling does not include flexible modes in bodies in the system. For a control application, if the closed-loop control bandwidth is significantly less than the bandwidth of the major flexible modes of the system, then the flexibility will not have a significant effect. The derivations assume continuous time; however, simulations and experiments use a controller running at discrete time intervals. When CT control systems are run at greater than ten times their closed-loop bandwidth, they approximate continuous time.

While friction is not included in the dynamic model, it can frequently be modeled and compensated for external to the CT control system. Computed-torque control theory per se does not address the issues of drive-train flexibility nor drive-train friction and backlash. Uhlik [33], Hollars [9] and others have investigated drive-train flexibility and Pfeffer and Khatib [23] and Hollars and Tilley [31] demonstrated that friction and backlash effects can be minimized by using a tight feedback controller wrapped around each joint.

This derivation uses Kane's [13] dynamical analysis concepts, techniques and notational conventions. A familiarity with this notation, presented at the end of the introduction, would be beneficial to the interested reader. Basic concepts, such as partial velocities, are introduced in the derivation as they are required.

2.2 Kinematics

Kinematics is the expression of the velocities and accelerations of points, and the angular velocities and accelerations of bodies. These quantities are essential for calculating dynamics terms, which are a function of accelerations, and are also required for specification of constraints (e.g. closed-chain constraints) and desired system behavior for control.

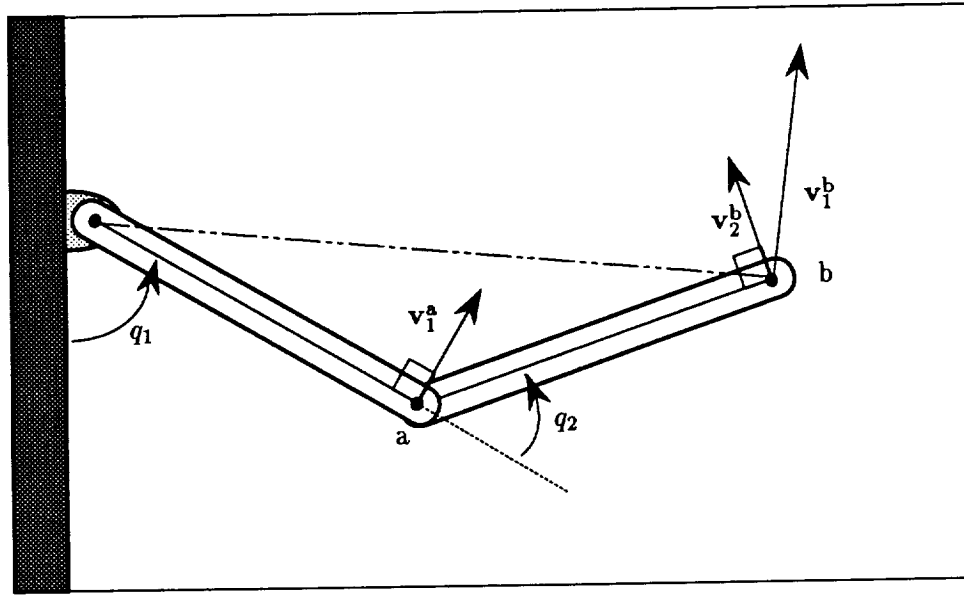


Figure 2.1: **Partial Velocities and Generalized Speeds**
The choice of generalized speeds (linear combinations of coordinate rates) determines the partial velocities. In this example, $u_1 = \dot{q}_1$, and $u_2 = \dot{q}_2$.

2.2.1 Velocities

One of the fundamental concepts underlying Kane's dynamics derivation is the relation between velocities of a point, its partial velocities, and the generalized speeds of the system. Velocities \mathbf{v} of points and angular velocities $\boldsymbol{\omega}$ of bodies in a system can be expressed in a Newtonian reference frame as linear combinations of the partial velocities, \mathbf{v}_r , and $\boldsymbol{\omega}_r$. While partial velocities are not exactly 'physical' quantities (they are a function of definitions made by the analyst), if intelligently chosen, they can have intuitive meaning. Figure 2.1 shows a two-link manipulator arm with endpoint velocity and partial velocities marked for the case that the generalized speeds are the derivatives of the joint angles. The partial velocities of the two points a and b for the two generalized speeds are denoted by $\mathbf{v}_1^a, \mathbf{v}_2^a$, and $\mathbf{v}_1^b, \mathbf{v}_2^b$.

In this analysis, it is assumed that the system is not undergoing unalterable motions due to external forces, hence the partial velocity residuals \mathbf{v}_i and $\boldsymbol{\omega}_i$ are zero. Therefore

the partial velocities and velocities are related as follows:

$$\begin{aligned} \mathbf{v}^i &= \sum_{r=1}^n \mathbf{v}_r^i u_r \\ \omega^j &= \sum_{r=1}^n \omega_r^j u_r \end{aligned} \quad (2.1)$$

where \mathbf{v}_r^i is the partial velocity of point i with respect to generalized speed u_r .

Once the generalized coordinates and speeds are selected for a dynamic system, the partial velocities and partial angular velocities for points and bodies in the system may be determined in the conventional manner. The general methods for determining them is left to the analyst and is completely general. At this stage of the analysis, no form or structure of these partial velocities can be, nor is, assumed. It turns out that partial velocities are very useful vector quantities – they will be used in almost all of the derivations in this thesis.

2.2.2 Accelerations

Acceleration expressions can be determined by differentiating velocity expressions. For the purposes of formulating equations of motion the acceleration expressions need to be sorted into those terms linear in generalized speed derivatives \dot{u} , and those not – the nonlinear terms. Rather than taking the derivative of the velocity expressions directly, it is advantageous to take the derivative of the partial velocity expressions previously shown in equation 2.1:

$$\mathbf{a}^i = \sum_{r=1}^n \mathbf{v}_r^i \dot{u}_r + \sum_{r=1}^n \dot{\mathbf{v}}_r^i u_r \quad (2.2)$$

$$\alpha^i = \sum_{r=1}^n \omega_r^i \dot{u}_r + \sum_{r=1}^n \dot{\omega}_r^i u_r \quad (2.3)$$

This automatically produces the terms linear in generalized speed derivatives (the first sum of equations 2.2 and 2.3) separately from those that are nonlinear (the second sum). An added benefit is that half of the result does not require the taking of derivatives: only the nonlinear terms require taking derivatives, and then only of partial velocity expressions. This result is used to formulate equations of motion in section 2.3 and the Jacobian matrix equation in section 3.2. The complexity of the acceleration expressions and the sorting of terms can be completely avoided this way.

2.3 Dynamics

Dynamics is a branch of mechanics that deals with forces and their relation to the motion of bodies in a system. Dynamic relations are used to formulate the system's equations of motion, which can be used to simulate system behavior, or help control it. Simulation is accomplished by solving for accelerations in the system, given a state consisting of position, orientation and velocity, and given actuator forces and/or torques. Robotic computed-torque control systems use the equations of motion to solve for actuator forces and/or torques, given system state and the desired joint accelerations. In this section, the equations of motion for systems of ν bodies are formalized in terms of the linear and angular momenta. Of particular interest is the way partial velocities and momenta are related, and the effect of this relation on a general expression for terms in the matrix equations of motion.

It is common to derive unconstrained dynamics expressions first, and then constrain them as required. This approach will also be followed in this presentation.

Equations of motion for robotic systems are often expressed [6] as

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau} \quad (2.4)$$

where the matrix \mathbf{M} is referred to as the mass (or inertia) matrix, $\ddot{\mathbf{q}}$ are the second derivatives of the system coordinates (typically joint angles), \mathbf{V} are the nonlinear terms, and $\boldsymbol{\tau}$ are the torques (or forces) at the joint actuators.

This analysis will develop a dual of this equation, in which the elements are similar but more generalized:

$$\mathbf{M}\dot{\mathbf{u}} = -\mathbf{N}\mathbf{u} + \mathbf{F} \quad (2.5)$$

An augmented form for constrained dynamic systems that resolves all the accelerations by including the constraints \mathbf{C} and constraint forces \mathbf{F}^C is

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{u}}_{1..n} \\ \mathbf{F}^C \end{bmatrix} = - \begin{bmatrix} \mathbf{N} \\ \dot{\mathbf{C}} \end{bmatrix} \mathbf{u}_{1..n} + \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix} \quad (2.6)$$

Both the unconstrained and constrained dynamics simulation equations are highly structured when built using partial velocities and their derivatives as basic building blocks.

Velocities, angular velocities, momenta and their derivatives in this derivation are with respect to a Newtonian reference frame¹.

Kane's [13] dynamics equations

$$F_r + F_r^* = 0 \quad (2.7)$$

describe the interaction of the general active forces F_r due to applied forces and torques and the generalized inertia forces F_r^* due to accelerations of masses. The generalized active forces are a function of applied forces F^j and the partial velocities v_r^{j*} of the point they are applied at, and the applied torques T^k and the partial angular velocities ω_r^k of the body on which they are applied.

$$F_r = \sum_{\substack{\text{applied} \\ \text{forces } j}} F^j \cdot v_r^{j*} + \sum_{\substack{\text{applied} \\ \text{torques } k}} T^k \cdot \omega_r^k \quad (2.8)$$

The generalized inertia forces are a function of the d'Alembert forces F^{j*} due to accelerations of masses and the partial velocities v_r^{j*} of their point of action, and the d'Alembert moments T^{k*} due to angular accelerations of bodies and the partial angular velocities ω_r^k of those bodies.

$$F_r^* = \sum_{\substack{\text{applied} \\ \text{forces } j}} F^{j*} \cdot v_r^{j*} + \sum_{\substack{\text{applied} \\ \text{torques } k}} T^{k*} \cdot \omega_r^k \quad (2.9)$$

The generalized inertial forces can be derived from the linear and angular momenta of the ν bodies in the system. First, the terms due to changes in linear momentum will be examined, then terms due to changes in angular momentum will be examined. The linear momentum of bodies L^i and the angular momentum of bodies $H^{i/i*}$ about their mass center i^* are expressed with respect to a Newtonian reference frame.

The linear momentum of body i is

$$\begin{aligned} L_{\text{Newton}}^i &= m_i v^{i*} \\ &\stackrel{2.1}{=} m^i \sum_{r=1}^n v_r^{i*} u_r \\ &= \sum_{r=1}^n m^i v_r^{i*} u_r \\ &= \sum_{r=1}^n L_r^i u_r \end{aligned} \quad (2.10)$$

¹The notation of which has been omitted for clarity.

where *partial linear momenta* is defined as

$$\mathbf{L}_r^i \stackrel{\Delta}{=} m_i \mathbf{v}_r^{i*} \quad (2.11)$$

The inertia (d'Alembert) force \mathbf{F}^{i*} caused by the acceleration of the center of mass of body i is:

$$\begin{aligned} \mathbf{F}^{i*} &= -\frac{d}{dt} \mathbf{L}^i \\ &\stackrel{2.11}{=} -\sum_{s=1}^n \dot{\mathbf{L}}_s^i u_s - \sum_{s=1}^n \mathbf{L}_s^i \dot{u}_s \end{aligned} \quad (2.12)$$

Its contribution to the generalized inertia forces is

$$\mathbf{F}^{i*} \cdot \mathbf{v}_{\Sigma_{11,2.12}}^i = -\sum_{s=1}^n \dot{\mathbf{L}}_s^i \cdot \mathbf{v}_r^{i*} u_s - \sum_{s=1}^n \mathbf{L}_s^i \cdot \mathbf{v}_r^{i*} \dot{u}_s \quad (2.13)$$

The contribution of the changes in angular momentum will now be examined. The angular momentum of body i is

$$\begin{aligned} \mathbf{H}^i &\stackrel{[13]}{=} \mathbf{I}^{i/i*} \boldsymbol{\omega}^i \\ &\stackrel{2.1}{=} \mathbf{I}^{i/i*} \sum_{r=1}^n \boldsymbol{\omega}_r^i u_r \\ &= \sum_{r=1}^n \mathbf{I}^{i/i*} \boldsymbol{\omega}_r^i u_r \\ &= \sum_{r=1}^n \mathbf{H}_r^i u_r \end{aligned} \quad (2.14)$$

where *partial angular momenta* is defined as

$$\mathbf{H}_r^i \stackrel{\Delta}{=} \mathbf{I}^{i/i*} \boldsymbol{\omega}_r^i \quad (2.15)$$

The inertia (d'Alembert) torque \mathbf{T}^{i*} caused by the angular acceleration of body i is

$$\begin{aligned} \mathbf{T}^{i*} &\stackrel{[13]}{=} -\frac{d}{dt} \mathbf{H}^i \\ &\stackrel{2.1}{=} -\sum_{s=1}^n \dot{\mathbf{H}}_s^i u_s - \sum_{s=1}^n \mathbf{H}_s^i \dot{u}_s \end{aligned} \quad (2.16)$$

Its contribution to the generalized inertia forces are

$$\mathbf{T}^{i*} \cdot \boldsymbol{\omega}_{\Sigma_{15,2.16}}^i = -\sum_{s=1}^n \dot{\mathbf{H}}_s^i \cdot \boldsymbol{\omega}_r^i u_s - \sum_{s=1}^n \mathbf{H}_s^i \cdot \boldsymbol{\omega}_r^i \dot{u}_s \quad (2.17)$$

The generalized inertia forces can be then be expressed as

$$F_{r_{2.13,2.17}}^* = - \sum_{i=1}^{\nu} \left(\sum_{s=1}^n \dot{L}_s^i \cdot \mathbf{v}_r^{i*} u_s + \sum_{s=1}^n L_s^i \cdot \mathbf{v}_r^{i*} \dot{u}_s \right) - \sum_{i=1}^{\nu} \left(\sum_{s=1}^n \dot{H}_s^i \cdot \omega_r^i u_s + \sum_{s=1}^n H_s^i \cdot \omega_r^i \dot{u}_s \right) \quad (2.18)$$

The generalized inertia forces have two components, one of which, $\mathbf{F}_r^{\mathcal{M}}$, is linear in the derivatives of the generalized speeds.

$$F_r^* \stackrel{\Delta}{=}_{2.18} \mathbf{F}_r^{\mathcal{M}} + \mathbf{F}_r^{\mathcal{N}} \quad (2.19)$$

These two components can be separated and expressed as

$$F_r^{\mathcal{M}} \stackrel{\Delta}{=}_{2.19} - \sum_{i=1}^{\nu} \left[\sum_{s=1}^n L_s^i \cdot \mathbf{v}_r^{i*} \dot{u}_s - \sum_{s=1}^n H_s^i \cdot \omega_r^i \dot{u}_s \right] \quad (2.20)$$

$$F_r^{\mathcal{N}} \stackrel{\Delta}{=}_{2.19} - \sum_{i=1}^{\nu} \left[\sum_{s=1}^n \dot{L}_s^i \cdot \mathbf{v}_r^{i*} u_s - \sum_{s=1}^n \dot{H}_s^i \cdot \omega_r^i u_s \right] \quad (2.21)$$

The inertia and momentum scalars which make up the mass matrix \mathbf{M} and those that make up the nonlinear coupling matrix \mathbf{N} can then be evaluated as follows:

$$m_{rs} \stackrel{\Delta}{=}_{2.18,2.20} \sum_{i=1}^{\nu} m_i \mathbf{v}_s^{i*} \cdot \mathbf{v}_r^{i*} + \left(\mathbf{I}^{i/i*} \omega_s^i \right) \cdot \omega_r^i \quad (2.22)$$

$$n_{rs} \stackrel{\Delta}{=}_{2.18,2.21} \sum_{i=1}^{\nu} m_i \dot{\mathbf{v}}_s^{i*} \cdot \mathbf{v}_r^{i*} + \left(\omega^i \times \mathbf{I}^{i/i*} \omega_s^i \right) \cdot \omega_r^i \quad (2.23)$$

The two components $\mathbf{F}^{\mathcal{M}}$ and $\mathbf{F}^{\mathcal{N}}$ can be expressed in terms of matrix vector products:

$$\mathbf{F}_r^{\mathcal{M}} = - \sum_{s=1}^n m_{rs} \dot{u}_s \quad (2.24)$$

$$\mathbf{F}_r^{\mathcal{N}} = - \sum_{s=1}^n n_{rs} u_s \quad (2.25)$$

The mass matrix, comprised of terms m_{rs} , and a nonlinear coupling matrix, comprised of terms n_{rs} , can then be used to express the equations of motion of the system as:

$$\begin{aligned} \mathbf{M} \dot{\mathbf{u}} &= -\mathbf{N} \mathbf{u} + \mathbf{F} \\ \tau &= \mathbf{W}^{-1} \mathbf{F} \end{aligned} \quad (2.26)$$

where the matrix \mathbf{W} maps generalized speeds into the derivatives of the generalized coordinates (joint rates).

$$\dot{\mathbf{q}}_r = \sum_{s=1}^n \mathbf{W}_{rs} u_s \quad (2.27)$$

The inverse of the matrix \mathbf{W} is typically trivial and time-invariant, and depends on the analyst's choice of generalized speeds. The generalized active force vector \mathbf{F} (of \mathbf{F}_r) accounts for the effects of external forces applied to points and torques applied to bodies in the system:

$$\mathbf{F}_r \triangleq_{[13]} \sum_{\text{All external forces}} \mathbf{v}_r^j \cdot \mathbf{F}^j + \sum_{\text{All external torques}} \boldsymbol{\omega}_r^i \cdot \mathbf{T}^i \quad (2.28)$$

2.4 Nonholonomic Motion Constraints

In a dynamic system with nonholonomic constraints, the generalized speeds $u_{1..n}$ are not independent, rather, one or more are dependent on the rest. This comes about due to a motion constraint : velocities or angular velocities are constrained. An example of this is shown in figure 2.2, where the closed kinematic chain has fewer degrees of freedom than the system would have if it were cut. In this section constraint equations will be formulated and dynamic equations that allow simulation will be presented. The resultant simulation matrix equation is very simple to formulate and requires no special techniques to solve.

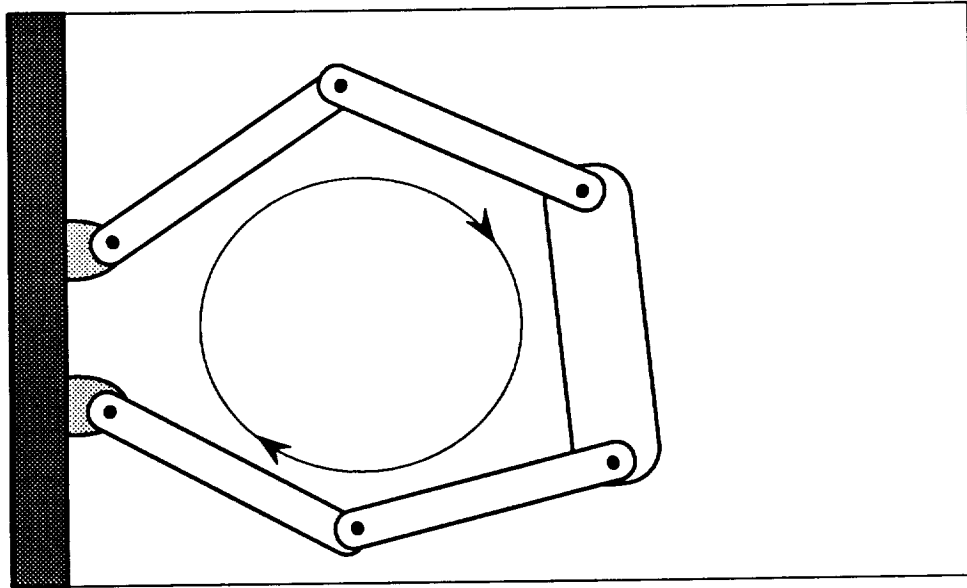


Figure 2.2:

A Closed Kinematic Chain

A closed kinematic chain arises when a kinematic chain has closed topological loops.

2.4.1 Constraint Equations

Constraints in motion can be one of three types: the velocity \mathbf{v}^P of a point is zero, the \mathbf{v}^P velocity of a point is the same as the velocity $\mathbf{v}^{P'}$ of another point, or a velocity is constrained to some unalterable value². This analysis is applicable to systems that have one or more constraints in linear and/or angular velocity. A velocity constraint is expressed easily as:

$$\mathbf{v}^P = \mathbf{v}^{P'} \quad (2.29)$$

where $\mathbf{v}^{P'} = 0$ if the velocity \mathbf{v}^P is constrained to zero. A simple velocity constraint expression will be formally defined, and then decomposed into a set of equations of the order of the constraint (2 for 2D systems, 3 for 3D systems). We define a *constraint velocity* as:

$$\mathbf{c}^P = \mathbf{v}^P - \mathbf{v}^{P'} \quad (2.30)$$

where

$$\mathbf{c}^P = 0$$

These constraint velocities can be expressed using partial velocities as follows:

$$\begin{aligned} \mathbf{c}^P &= \sum_{r=1}^n \mathbf{v}_r^P u_r - \sum_{r=1}^n \mathbf{v}_r^{P'} u_r \\ &= \sum_{r=1}^n \mathbf{c}_r^P u_r \end{aligned} \quad (2.31)$$

where *partial constraint velocities*³ are defined as

$$\mathbf{c}_r^P \triangleq \mathbf{v}_r^P - \mathbf{v}_r^{P'} \quad (2.32)$$

Velocities of points in 3D can be expressed in terms of speeds along some established inertial x,y and z directions, for example, along inertial unit vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$:

$$\mathbf{c}^P \cdot \hat{\mathbf{x}} = \sum_{r=1}^n \mathbf{c}_r^P \cdot \hat{\mathbf{x}} u_r$$

²Such 'Forced motion' constraints introduce partial velocity residuals \mathbf{v}_t and ω_t , and are not covered in this analysis.

³Even although the *constraint velocity* is zero, the *partial constraint velocities* are not.

$$\begin{aligned} \mathbf{c}^p \cdot \hat{\mathbf{y}} &= \sum_{r=1}^n \mathbf{c}_r^p \cdot \hat{\mathbf{y}} u_r \\ \mathbf{c}^p \cdot \hat{\mathbf{z}} &= \sum_{r=1}^n \mathbf{c}_r^p \cdot \hat{\mathbf{z}} u_r \end{aligned}$$

and express the constraint on generalized speeds⁴ as a matrix equation

$$\mathbf{C}u \triangleq 0 \quad (2.33)$$

where the elements c_{sr} of the constraint matrix \mathbf{C} are:

$$\begin{aligned} c_{1r}^p &= \mathbf{c}_r^p \cdot \hat{\mathbf{x}} \\ c_{2r}^p &= \mathbf{c}_r^p \cdot \hat{\mathbf{y}} \\ c_{3r}^p &= \mathbf{c}_r^p \cdot \hat{\mathbf{z}} \end{aligned}$$

The constraint equation can also be used to express constraints in terms of the derivatives of the generalized speeds. This kind of constraint is useful when solving for these accelerations in either dynamics simulations or control to guarantee that the system experiences motions (or is commanded to move) in a manner consistent with the constraint:

$$\mathbf{C}\dot{u} + \dot{\mathbf{C}}u = 0 \quad (2.34)$$

The elements \dot{c}_{sr} of the derivative of the constraint matrix $\dot{\mathbf{C}}$ are simply measures of the time derivatives of the *partial constraint velocities* along the inertial basis:

$$\begin{aligned} \dot{c}_{1r}^p &= \dot{\mathbf{c}}_r^p \cdot \hat{\mathbf{x}} \\ \dot{c}_{2r}^p &= \dot{\mathbf{c}}_r^p \cdot \hat{\mathbf{y}} \\ \dot{c}_{3r}^p &= \dot{\mathbf{c}}_r^p \cdot \hat{\mathbf{z}} \end{aligned} \quad (2.35)$$

The derivatives of the *partial constraint velocities* are the derivatives of the partial velocities of constraint's associated points.

$$\begin{aligned} \dot{\mathbf{c}}_r &\triangleq \frac{d}{dt} \mathbf{c}_r \\ &= \dot{\mathbf{v}}_r^p - \dot{\mathbf{v}}_r^{p'} \end{aligned} \quad (2.36)$$

⁴If $u_r = \dot{q}_r$, then this constraint is explicitly on joint rates.

2.4.2 Constraining the Equations of Motion

In this section the unconstrained equations of motion are modified by the addition of constraints. These modifications are due to the forces and/or moments introduced by the constraint, which may or may not be of interest. These forces and/or moments are non-contributing to the dynamics: they can be left out of the dynamics simulation equations because they perform no work on the system. This is evident in Kane's representation of constrained equations of motion:

$$\tilde{\mathbf{F}}_r + \tilde{\mathbf{F}}_{r=1..p}^* = 0 \quad (2.37)$$

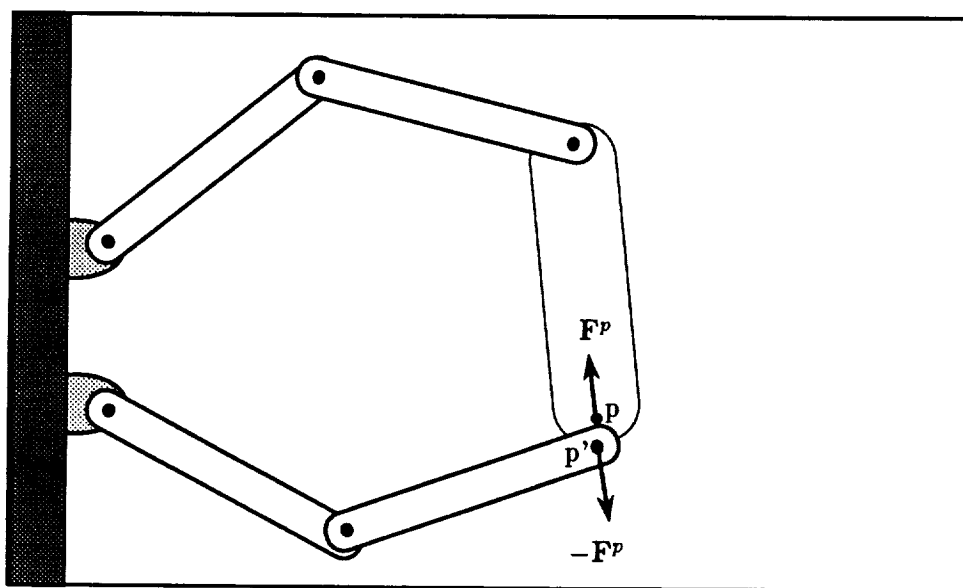


Figure 2.3:

A Constraint Force

A constraint force and/or moment arises out of a motion constraint.

The number of degrees of freedom in this dynamic system are p , where $p = n - c$ from an unconstrained system that has n degrees of freedom with c constraint equations.

It is instructive to express the constrained equations of motion using the constraints in order to see just how simply they can be expressed. The simplest way to ensure that the accelerations in the system are consistent with the constraint is to augment the open-chain dynamics with the constraint forces and the constraint equations described in the previous

section. Figure 2.3 shows a typical constraint force due to a motion constraint at point p . The constraint force of \mathbf{F}^p acts on point p , and a corresponding force $-\mathbf{F}^p$ acts on the point p' , to which the motion of point p is constrained (a similar case exists in the event of constraint torques). The constraint forces and torques can be added to the open-chain dynamics by expressing them as generalized forces.

$$\begin{aligned}
 \mathbf{F}_r^{\text{constraint}} &= \sum_{\substack{\text{Applied} \\ \text{forces } j}} \mathbf{F}^j \cdot \mathbf{v}_r^{j*} + \sum_{\substack{\text{Applied} \\ \text{torques } k}} \mathbf{T}^k \cdot \boldsymbol{\omega}_r^k \\
 &= \sum_{\substack{\text{Applied} \\ \text{forces } j}} \mathbf{F}^j \cdot \mathbf{c}_r^{j*} + \sum_{\substack{\text{Applied} \\ \text{torques } k}} \mathbf{T}^k \cdot \mathbf{c}_r^k \\
 &\stackrel{2.33}{=} \mathbf{C}^T \mathbf{F}^C
 \end{aligned} \tag{2.38}$$

The unconstrained equations of motion can be modified by the addition of the constraint forces and moments and the motion constraint terms, to yield the constrained equations of motion. The inclusion of c motion constraint equations increases the number of equations to solve simultaneously from n to $n + c$.

$$\mathbf{F}_r^{\text{constraint}} + \mathbf{F}_r + \mathbf{F}_r^* = 0 \tag{2.39}$$

$$\mathbf{C}\dot{\mathbf{u}} + \dot{\mathbf{C}}\mathbf{u} \stackrel{2.34}{=} 0 \tag{2.40}$$

In matrix form, the constrained equations of motion can be expressed as follows:

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{u}}_{1..n} \\ \mathbf{F}^C \end{bmatrix} \stackrel{2.6}{=} \begin{bmatrix} \mathbf{N} \\ \dot{\mathbf{C}} \end{bmatrix} \mathbf{u}_{1..n} + \begin{bmatrix} \mathbf{F} \\ \mathbf{0} \end{bmatrix}$$

This formulation shows that it is possible to solve for both the constraint forces and the system accelerations simultaneously. Alternatively, it is possible to specify both system accelerations *and* internal forces if performing inverse dynamics for control.

2.4.3 Ensuring Constraints Hold in Simulation

One of the problems with dynamic simulations of constrained systems is ensuring consistency in the state. It is generally not possible to start the simulation with a *perfectly* consistent state, and initial inconsistencies may grow.

While the equations of motion for simulating constrained systems (equation 2.6) ensure consistent *accelerations* given a consistent state, they will *not* correct any inconsistencies in the state.

It turns out that a simple modification of these equations of motion can take care of this problem. The constraints can be guaranteed by using a numerical relaxation method. Consider, for example, a motion constraint between two points, p and p' . It is possible for a position error of $\mathbf{r}^p - \mathbf{r}^{p'}$ to develop, as well as velocity errors of $\mathbf{v}^p - \mathbf{v}^{p'}$. These points can be made coincident in space if their accelerations are modified as follows (like a spring-dashpot arrangement):

$$\mathbf{a}^p - \mathbf{a}^{p'} = -K_p(\mathbf{r}^p - \mathbf{r}^{p'}) - K_v(\mathbf{v}^p - \mathbf{v}^{p'}) \quad (2.41)$$

The gain values K_p and K_v determine the numerical relaxation characteristics, and should be adapted to the simulation step size. The choice of these gains are similar to the choice of position and velocity feedback gains in a second-order, discrete-time system – if they are chosen incorrectly, the simulation can be *unstable*. A rule of thumb is to choose them to have a bandwidth ω_n of about 10 times less than the simulation rate, with a large amount of damping $\zeta = 1..2$. To ensure quick convergence to a consistent state at simulation startup, make the timestep t very small and the gains K_p and K_v large – then go to the regular timestep and much lower gains for the rest of the simulation.

$$K_p = \omega_n^2 \quad (2.42)$$

$$K_v = \frac{2\omega_n}{\zeta} \quad (2.43)$$

The relaxation equation can be recast as

$$\epsilon^p \stackrel{\Delta}{=}_{2.41} -K_p(\mathbf{r}^p - \mathbf{r}^{p'}) - K_v(\mathbf{v}^p - \mathbf{v}^{p'}) \quad (2.44)$$

and then merged with the constrained equations of motion

$$\begin{bmatrix} \mathbf{M} & \mathbf{C}^T \\ \mathbf{C} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \dot{\mathbf{u}}_{1..n} \\ \mathbf{F}^C \end{bmatrix} \stackrel{2.6, 2.44}{=} \begin{bmatrix} \mathbf{N} \\ \dot{\mathbf{C}} \end{bmatrix} \mathbf{u}_{1..n} + \begin{bmatrix} \mathbf{F} \\ (\epsilon^p)^* \end{bmatrix} \quad (2.45)$$

so that a new set of equations of motion arise. These equations of motion for simulation of dynamic systems with constraints make use of numerical relaxation techniques to ensure constraints are met. They can be used to remove initial state inconsistencies, or to prevent the buildup of numerical error.

2.4.4 Reduced-Order Equations of Motion

Since constraints reduce the number of degrees of freedom in a dynamic system, the equations of motion are usually expressed in reduced-order form. The procedure for producing such a reduced-order form from the augmented form of equation 2.6 is presented here. The disadvantages of using this form, as opposed to the augmented form of equation 2.45 are discussed as well.

When a system is constrained, generalized speeds are no longer completely independent. Kane expresses the c constraint equations that arise as:

$$u_r = \sum_{s=1}^p A_{rs} u_s + B_r \quad (2.46)$$

Where the first p generalized speeds are taken as the independent ones. This analysis has assumed conditions that result in partial velocity residuals \mathbf{v}_t to be zero: as a consequence, the \mathbf{B}_r terms are also zero. The constraining matrix \mathbf{A} can be determined by partitioning the motion constraint matrix into two parts: the first part is $c \times p$ and the second $c \times c$:

$$\mathbf{C} = \left[\mathbf{C}_{c \times p}^u : \mathbf{C}_{c \times c}^c \right]_{c \times n} \quad (2.47)$$

$$\mathbf{C} \mathbf{u} = 0 \quad (2.48)$$

$$\mathbf{C}^u u_{1..p} + \mathbf{C}^c u_{(p+1)..n} = 0 \quad (2.49)$$

$$u_{(p+1)..n} = -(\mathbf{C}^c)^{-1} \mathbf{C}^u u_{1..p} \quad (2.50)$$

The constraining matrix can then be expressed using these two parts:

$$\mathbf{A} = -(\mathbf{C}^c)^{-1} \mathbf{C}^u \quad (2.51)$$

The generalized forces and generalized inertia forces for the nonholonomic system are

$$\tilde{\mathbf{F}}_r = \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{F}_r \quad (2.52)$$

$$\tilde{\mathbf{F}}_r^* = \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{F}_r^* \quad (2.53)$$

For this simple nonholonomic system possessing p degrees of freedom the reduced mass matrix, nonlinear terms, and the force distribution matrix are

$$\begin{aligned}\tilde{\mathbf{M}} &= \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{M} \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix} \\ \tilde{\mathbf{N}} &= \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{N} \\ \tilde{\mathbf{W}} &= \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{W}\end{aligned}\tag{2.54}$$

In this minimal order system, generalized accelerations can be solved for as follows:

$$\dot{u}_{1..p} = \tilde{\mathbf{M}}^{-1}(-\tilde{\mathbf{N}}u_{1..n} + \tilde{\mathbf{W}}\tau)\tag{2.55}$$

$$(2.56)$$

The disadvantages of using this form (equation 2.55) for simulation, as opposed to the augmented form (equation 2.45), are the following: (1) a fair amount of computation is required to reduce the dynamics matrices to minimal order form, and (2) once system accelerations are solved for, it is still necessary to back out the rest of the state. The remaining parts of the state can be determined using constraint relations and inverse kinematics – a complicated and frequently iterative process. This can slow down the simulation process significantly.

2.5 Summary

In this chapter, kinematics and dynamics relations applicable to systems of rigid bodies were presented. A new method for formulating equations of motion for simulation of constrained dynamic systems was presented. These simulation equations (equations 2.45) ensure that constrained systems converge on a consistent state despite small initial state errors, via numerical relaxation. The buildup of numerical error (a violation of constraints) is also prevented.

Chapter 3

Computed Torque Control

Most commercial robot control systems are based on joint-by-joint position and rate feedback (PD) control. In a free-flying robot, however, the manipulator arm joint angles *do not* uniquely determine the endpoint position of manipulator endpoints, nor of manipulator end-effector orientation. These positions and orientations are functions of the robot base's position and orientation as well. It is not possible to specify manipulator endpoint trajectories in terms of only manipulator joint angles, such as for joint PD control.

Furthermore, cooperating arm manipulators have motion constraints that force angles and angle rates to be related through constraint relations: specifying inexact angles and rate to a joint PD controller can result in large tensile or compressive forces on a manipulated object. The computed-torque (CT) control method, also known as the resolved-acceleration method is better suited to such systems. A CT controller can compensate for the highly nonlinear dynamics of a cooperating-arm robot, and allows direct specification of desired endpoint/object accelerations. Using an endpoint feedback control law with CT control, it is possible to achieve precise, high-performance manipulator response.

In this chapter the CT control technique is extended to free-flying and constrained dynamic systems, while maintaining a standard form. This is accomplished by augmenting the Jacobian matrix with momentum and/or dynamic constraints. In a system that has dynamic constraints, this formulation allows the use of unconstrained inverse dynamics - a distinct advantage over current formulations, which require the use of expensive-to-compute constrained inverse dynamics.

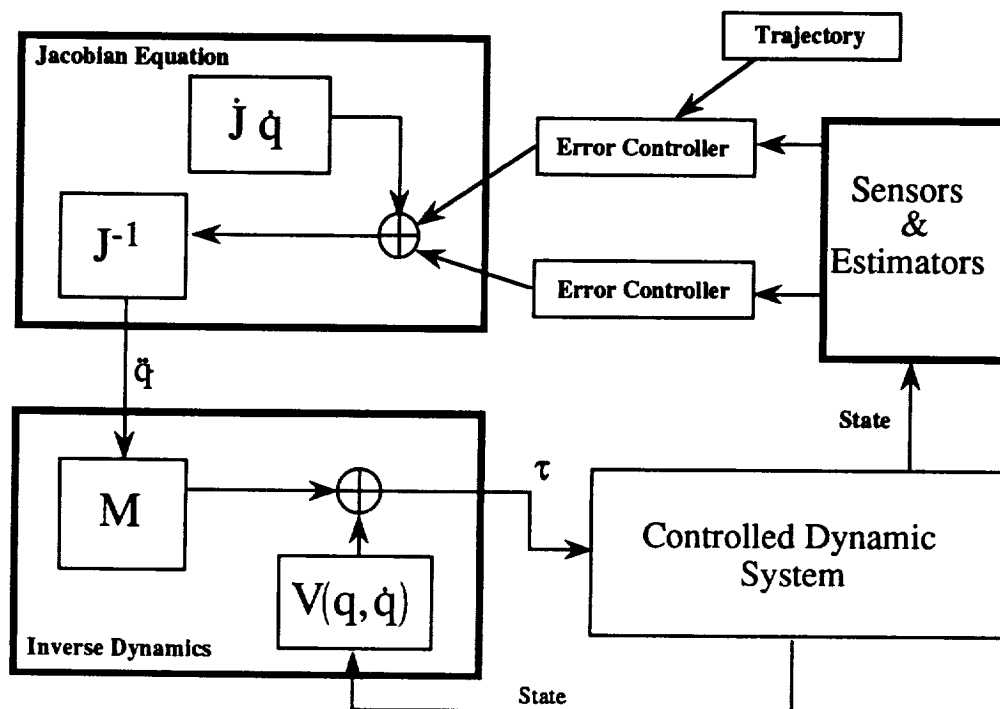


Figure 3.1: **Block Diagram of Computed-Torque Control**

Decoupling linearized control of highly non-linear systems can be achieved with an accurate dynamic model. The two major components are the Jacobian section and the inverse dynamics section. Error controllers specify the desired error dynamics of chosen controlled items. If the full system state cannot be measured estimators are required.

3.1 Components of a Computed Torque Controller

A basic computed-torque controller, as shown in figure 3.1 consists of seven parts. The trajectory section provides desired trajectories, which can specify smooth motions, for the quantities under control. The error controllers, one for each quantity under control, use knowledge of the system state and a control law to bring errors to zero. The Jacobian equation translates commands in one space (that of the quantities under control) into generalized accelerations. The Jacobian matrix needs to be inverted to accomplish this. The generalized accelerations are derivatives of the system generalized speeds, which can be chosen by the analyst. They are generally chosen to be the joint angle rates. Once the generalized accelerations are known, the inverse dynamics process calculates the set of manipulator joint torques using knowledge of the mass distribution in the dynamic system,

its state, and the endpoint or internal forces, including internal friction, if modeled. The computed torques can be delivered directly by the actuators, or via tight torque-control loops wrapped around each actuator.

The plant – the dynamic system – will then react to these applied torques (and forces). Chances are the plant is not exactly modeled, so there will be some error. This error is reduced by closing the loop: feeding back the error between desired and actual motion. The error controllers rely on knowledge of the current value of the quantities of interest, commonly provided by sensors. The linearizing and decoupling aspects of computed-torque control rely on accurate knowledge of the state. Estimators may be necessary to provide an estimate of full system state if sufficient sensor measurements are not available, or can provide smoother estimates of system state given an accurate model of the plant and noisy signals.

This analysis will concern itself with the Jacobian and inverse dynamics: they are what makes computed-torque control both decoupling and linearizing for highly nonlinear systems.

There are two equations that implement the Jacobian and the inverse dynamics of computed-torque control. The Jacobian equation allows specification of desired system behavior, such as endpoint acceleration, body angular acceleration, joint acceleration, etc. The inverse dynamics equation uses joint accelerations computed in the first equation to solve for joint torques.

$$\begin{aligned}\ddot{q} &= \mathbf{J}^{-1}(\mathbf{a}_{des}^{endpoint} - \dot{\mathbf{J}} \dot{q}) \\ \tau &= \mathbf{M}\ddot{q} - V(q, \dot{q})\end{aligned}$$

In this analysis, a more generalized version of the Jacobian matrix equation will be derived to allow computed-torque control of complex dynamic systems. This equation involves the construction of an augmented Jacobian matrix \mathcal{J} .

$$\dot{u}_{3.33} = \mathcal{J}^{-1} \left(\begin{bmatrix} \mathbf{a}_{des}^{endpoint} \\ \vdots \end{bmatrix} - \dot{\mathcal{J}} u \right)$$

The solution to this Jacobian matrix equation is used by the generalized version of the (unconstrained) dynamics equation, developed earlier in section 2.3, to determine the joint

torques.

$$\mathbf{F} = \mathbf{M}\ddot{\mathbf{u}} + \mathbf{N}\mathbf{u}$$

$$\boldsymbol{\tau} = \mathbf{W}^{-1}\mathbf{F}$$

This chapter has two sections corresponding to these two equations. The first section presents a general formulation of the Jacobian matrix and augmentation equations that extend CT techniques to free-flying robots and dynamically constrained systems such as cooperative-arm manipulators. The second section will examine the inverse dynamics matrix equation, particularly in light of using an augmented Jacobian to solve for accelerations.

3.2 The Jacobian Matrix

A basic manipulator Jacobian as described by Craig [6] and Khatib [15] expresses manipulator endpoint speeds measured in a coordinate system as a function of the robot arm joint angles. In equation form this is expressed as

$$\mathbf{v}_{\text{endpoints}} = \mathbf{J}\dot{\mathbf{q}}$$

where \mathbf{v} is a vector of the speeds of the endpoints, and $\dot{\mathbf{q}}$ are the derivatives of the manipulator joint angles. This is fine when the number of degrees of freedom of the manipulator is the same as the number of degrees of freedom of the endpoints: the Jacobian matrix will be square.

Two cases where this is not so are free-flying manipulators and closed-chain manipulators. Free-flying manipulators have additional degrees of freedom – three in translation and three in rotation. In closed-chain manipulators such as cooperating-arm systems, the manipulated body has fewer degrees of freedom than the unconstrained dynamics – but the same number of degrees of freedom as the constrained system.

In order to use the computed-torque control technique, the Jacobian must be invertible. It must therefore be square. The process of making a Jacobian square by augmenting it with additional equations will be presented in this section. This augmented Jacobian will be depicted as \mathcal{J} . It allows specification not only of control in traditional operational space

(i.e. position, orientation) via \mathbf{J} , but also of momentum via \mathcal{J}^L and \mathcal{J}^H and systems with dynamic constraints via \mathcal{J}^C :

$$\mathcal{J} = \begin{bmatrix} \mathbf{J} \\ \mathcal{J}^L \\ \mathcal{J}^H \\ \mathcal{J}^C \end{bmatrix} \quad (3.1)$$

The first part of this augmented Jacobian to be examined will be the manipulator endpoint Jacobian \mathbf{J} . It will be recast in terms of generalized speeds and formulated with partial velocities. Next, it will be shown that momentum can be controlled via momentum augmentation equations $\mathcal{J}^L, \mathcal{J}^H$. Finally, it will be shown that motion constraints can be used as augmentation equations \mathcal{J}^C so that it is possible to control constrained systems and use unconstrained inverse dynamics to solve for torques. This new approach to forming an augmented Jacobian Matrix combined with unconstrained inverse dynamics is an elegant and simple solution to the otherwise complex problem of controlling free-flying or constrained systems.

3.2.1 Desired Acceleration Specification

A basic Jacobian, as defined by Craig and Khatib, is a matrix that relates manipulator endpoint velocities to joint rates. This matrix can be inverted in order to solve for joint rates in terms of endpoint velocities. The resolved-rate control technique uses this approach to get joint rate commands to joint-rate control systems. Umetani and Yoshida [36] have demonstrated such a technique on a free-flying robot.

Resolved rate control, however, does not lend itself well to force control, which is important for cooperative manipulation and other interactions with the environment¹. Resolved-rate control techniques assume that joint-based control systems can adequately reduce the dynamic coupling effects.

The basis for the computed-torque (or resolved acceleration) control scheme is to determine the desired joint accelerations from the desired endpoint accelerations using the

¹ Resolved rate control is computationally somewhat simpler, and can use analog control loops around joints to control their rates. For a slightly higher cost in computation, resolved acceleration can use analog control loops around joints to control torques.

Jacobian. While the analysis presented covers more than just endpoint accelerations, this first section shows the case where endpoint accelerations are the only control objectives. It will be shown that the Jacobian can be formulated using partial velocities and inertial basis vectors. A more generalized manipulator Jacobian expressed using generalized speeds² is:

$$\mathbf{v}^{\text{endpoint}} \stackrel{3.7}{=} \mathbf{J} \mathbf{u}$$

The endpoint acceleration can then be expressed as:

$$\mathbf{a}^{\text{endpoint}} = \mathbf{J} \dot{\mathbf{u}} + \dot{\mathbf{J}} \mathbf{u} \quad (3.2)$$

and the joint accelerations can be solved for by rearranging these equations:

$$\dot{\mathbf{u}}_{[6],[15]} = \mathbf{J}^{-1}(\mathbf{a}^{\text{endpoint}} - \dot{\mathbf{J}} \mathbf{u}) \quad (3.3)$$

The Jacobian matrix's components can be formulated using the partial velocities and partial angular velocities of the endpoint(s) of the manipulator(s) in the system. An endpoint's velocity, like any point in the dynamic system, can be expressed in terms of its partial velocities:

$$\mathbf{v}^{\text{endpoint}} = \sum_{r=1}^n \mathbf{v}_r^{\text{endpoint}} u_r \quad (3.4)$$

and therefore 3D endpoint velocity can be expressed in terms of speeds along some established inertial unit vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$:

$$\begin{aligned} \mathbf{v}^{\text{endpoint}} \cdot \hat{\mathbf{x}} &= \sum_{r=1}^n \mathbf{v}_r^{\text{endpoint}} \cdot \hat{\mathbf{x}} u_r \\ \mathbf{v}^{\text{endpoint}} \cdot \hat{\mathbf{y}} &= \sum_{r=1}^n \mathbf{v}_r^{\text{endpoint}} \cdot \hat{\mathbf{y}} u_r \\ \mathbf{v}^{\text{endpoint}} \cdot \hat{\mathbf{z}} &\stackrel{3.4}{=} \sum_{r=1}^n \mathbf{v}_r^{\text{endpoint}} \cdot \hat{\mathbf{z}} u_r \end{aligned} \quad (3.5)$$

the elements of the Jacobian due to an endpoint's velocity, \mathbf{J} , are therefore:

$$\begin{aligned} j_{1r} &= \mathbf{v}_r^{\text{endpoint}} \cdot \hat{\mathbf{x}} \\ j_{2r} &= \mathbf{v}_r^{\text{endpoint}} \cdot \hat{\mathbf{y}} \\ j_{3r} &\stackrel{3.5}{=} \mathbf{v}_r^{\text{endpoint}} \cdot \hat{\mathbf{z}} \end{aligned} \quad (3.6)$$

²If one chooses $\mathbf{u} \triangleq \dot{\mathbf{q}}$ then this is the standard Jacobian. If not, it becomes a more generalized Jacobian. The theory is valid for either case.

and the velocity of the point can be determined from the matrix equation

$$\mathbf{v}_{3.4,3.6} = \mathbf{J}u \quad (3.7)$$

As shown above, desired endpoint accelerations can be expressed in terms of the Jacobian, its derivative, and the generalized speeds and their derivatives. The derivatives of the elements of the Jacobian can also be determined from the partial velocities:

$$\begin{aligned} \dot{j}_{1r} &= \dot{\mathbf{v}}_r^{\text{endpoint}} \cdot \hat{\mathbf{x}} \\ \dot{j}_{2r} &= \dot{\mathbf{v}}_r^{\text{endpoint}} \cdot \hat{\mathbf{y}} \\ \dot{j}_{3r} &= \dot{\mathbf{v}}_r^{\text{endpoint}} \cdot \hat{\mathbf{z}} \end{aligned} \quad (3.8)$$

where the derivatives, taken in a Newtonian reference frame, of the partial velocities are

$$\dot{\mathbf{v}}_r^{\text{endpoint}} \triangleq \frac{d^N}{dt} \mathbf{v}_r^{\text{endpoint}} \quad (3.9)$$

These derivatives of partial velocities can be calculated from partial velocities and the angular velocity of the body (or frame) that the partial velocity vectors are based in.

Endpoint acceleration control specification can be expressed in terms of the Jacobian, its derivative, and the generalized speeds and their derivatives:

$$\mathbf{a}^{\text{endpoint}}_{3.7,3.8} = \mathbf{J}\dot{u}_{1..n} + \dot{\mathbf{J}}u_{1..n}$$

This completes the formal description of the Jacobian elements for desired accelerations. Note that desired angular accelerations can be treated in an identical manner, allowing body angular acceleration specification.

3.3 Jacobian Augmentation Equations

The Jacobian can be augmented with additional terms in order to cope with more complex manipulator systems. Past research into operational-space (computed-torque) control has conditioned people to thinking of a Jacobian matrix as something exclusively for dealing with manipulator endpoint velocities and angular velocities. In fact, a Jacobian or augmented Jacobian matrix³ can include momentum or motion constraints in the control

³By definition, a Jacobian is a generalized derivative. Properly, one should refer to the matrix as an augmented Jacobian matrix if some of the terms are not strictly derivatives (eg. Angular momentum).

objectives. Many additional alternative augmentation terms could be conceived: for example, handling redundancy. The material covered in this section has been previously published by the author in [2] and [17].

3.3.1 Momentum Control

If a system \mathcal{S} is free-flying it will possess 6 more degrees of freedom in 3D than when fixed to the ground, since it is free to translate and rotate. It is not possible to formulate a square manipulator-endpoint Jacobian for such systems, because the manipulator endpoint has fewer degrees of freedom than the robot's dynamic system. NASA's envisioned Orbital Maneuvering Vehicle is an example of such a system. These extra degrees of freedom need to be dealt with.

Base accelerations of a free-flying robot could be some of the control objectives, and thrusters could be used to provide necessary impulse to achieve them. This is not an desirable solution, however: every time a manipulator arm is moved, thruster gas would be expelled to provide a reaction force. Alternatively, the robot body can absorb these reactions, particularly if they have zero bias. This methodology makes it possible to control the manipulator and make minimal use of thrusters.

Alexander [1] has shown that a combination of the Jacobian matrix and a partitioned mass matrix can result in a solvable system. Constraining equations were taken directly from the mass matrix to partition and solve an otherwise redundant solution set. This method was not a viable solution in the general case, however, because dynamic equations involving mass matrix partitions may contain actuator torques, which are unknowns prior to solution.

Umetani and Yoshida [36] demonstrated that momentum relations – integrals of equations of motion – could be used to reduce in rank and solve the redundant system. Another approach taken to deal with the same problem was the definition of a Virtual Manipulator, by Vafa and Dubowski [7], so that a rederivation of dynamics terms could be made using knowledge of the linear momentum and the constraint of zero angular momentum. These formulations either constrain system momenta [7] [36] or require a separate external momentum control system [1].

A more general solution to this problem is to allow control of the extra degrees of freedom introduced by free flight within the scope of the CT controller. For example, it has already been shown by Ullman [34] that a robot body's position and orientation can be controlled to follow a trajectory. When dealing with computed-torque, however, it is possible to have quantities controlled that are *linear functions* of the generalized speeds of the system, not just of the coordinates (or rates) themselves. Therefore, the rate of change of the robot system's linear momentum can be specified: and a trajectory in momentum can be followed. An experimental demonstration of momentum control was done by Jasper [12], using a subset of a specific incarnation of the system dynamics equations to calculate joint accelerations, rather than a Jacobian per se.

This section will formalize the inclusion of momentum and dynamic constraints in the control objectives of a computed-torque controller. This will be accomplished by augmenting the manipulator Jacobian with linear and/or angular momenta equations. Inclusion of these relations can make a Jacobian full rank, and suitable for computed torque control.

3.3.2 Momentum Equations

First, the linear momentum, then the angular momentum of the system will be examined. The linear momentum \mathbf{L}^i of bodies and \mathbf{L}^S of the system are expressed with respect to a Newtonian reference frame. The angular momentum \mathbf{H}^{i/i^*} of bodies about their mass center i^* and the angular momentum \mathbf{H}^{S/S^*} of the system about the system's mass center S^* are expressed with respect to a Newtonian reference frame.

The linear momentum \mathbf{L}^S of a system of ν bodies is the sum of the linear momenta of each body i in the system, and can be expressed in terms of generalized speeds as follows:

$$\begin{aligned}
 \mathbf{L} &= \sum_{i=1}^{\nu} \mathbf{L}^i \\
 &= \sum_{i=1}^{\nu} m^i \mathbf{v}^{i^*} \\
 &= \sum_{i=1}^{\nu} m^i \sum_{r=1}^n \mathbf{v}_r^{i^*} u_r \\
 &= \sum_{i=1}^{\nu} \sum_{r=1}^n m^i \mathbf{v}_r^{i^*} u_r
 \end{aligned}$$

$$\begin{aligned}
&= \sum_{i=1}^{\nu} \sum_{r=1}^n \mathbf{L}_r^i u_r \\
&= \sum_{r=1}^n \mathbf{L}_r u_r
\end{aligned} \tag{3.10}$$

where the *partial linear momentum* \mathbf{L}_r^S of the system of ν bodies is defined by

$$\mathbf{L}_r^S \triangleq \sum_{i=1}^{\nu} m^i \mathbf{v}_r^{i*} \tag{3.11}$$

The *partial linear momenta* of the system can be formulated using the mass and center-of-mass partial velocity of each body in the system. The process of building an augmented Jacobian using these vector quantities is similar to the process used for the partial velocities discussed in the previous section, and will be examined after the angular momentum terms are derived.

The angular momentum $\mathbf{H}^{i/i*}$ of each body i about its center of mass is related to the body's *partial angular momenta* $\mathbf{H}_r^{i/i*}$ as follows:

$$\begin{aligned}
\mathbf{H}^i &= \mathbf{I}^{i/i*} \boldsymbol{\omega}^i \\
&\stackrel{2.15}{=} \sum_{s=1}^n \mathbf{H}_s^i u_s
\end{aligned}$$

The central angular momentum \mathbf{H}^{S/S^*} of a system of ν bodies about the system's center of mass point S^* at position \mathbf{r}^{cm} is⁴:

$$\begin{aligned}
\mathbf{H}^{S/S^*} &= \sum_{i=1}^{\nu} \mathbf{H}^i + \sum_{i=1}^{\nu} (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times m^i \mathbf{v}^{i*} \\
&= \sum_{i=1}^{\nu} (\mathbf{I}^{i/i*} \boldsymbol{\omega}^i + (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times m^i \mathbf{v}^{i*}) \\
&= \sum_{i=1}^{\nu} \left(\sum_{r=1}^n \mathbf{I}^{i/i*} \boldsymbol{\omega}_r^i u_r + \sum_{r=1}^n (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times m^i \mathbf{v}_r^{i*} u_r \right) \\
&= \sum_{i=1}^{\nu} \sum_{r=1}^n (\mathbf{H}_r^i u_r + (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times \mathbf{L}_r^i u_r) \\
&= \sum_{r=1}^n \mathbf{H}_r^{S/S^*} u_r
\end{aligned} \tag{3.12}$$

where the *partial angular momentum* \mathbf{H}_r^{S/S^*} of the system is defined as

$$\mathbf{H}_r^{S/S^*} \triangleq \sum_{i=1}^{\nu} (\mathbf{H}_r^i + (\mathbf{r}^{i*} - \mathbf{r}^{cm}) \times \mathbf{L}_r^i) \tag{3.13}$$

⁴See Kane [13] pp69, §6.3)

Jacobian augmentation equations can be set up which describe the relation between the momenta and the generalized speeds.

$$\mathbf{L}^S = \mathcal{J}^L u \quad (3.14)$$

$$\mathbf{H}^{S/S^*} = \mathcal{J}^H u \quad (3.15)$$

The elements of the Jacobian due to the linear and angular momenta are therefore:

$$\begin{aligned} \mathcal{J}_{1r}^L &= \mathbf{L}_r^S \cdot \hat{\mathbf{x}} \\ \mathcal{J}_{2r}^L &= \mathbf{L}_r^S \cdot \hat{\mathbf{y}} \\ \mathcal{J}_{3r}^L &= \mathbf{L}_r^S \cdot \hat{\mathbf{z}} \end{aligned} \quad (3.16)$$

and

$$\begin{aligned} \mathcal{J}_{1r}^H &= \mathbf{H}_r^{S/S^*} \cdot \hat{\mathbf{x}} \\ \mathcal{J}_{2r}^H &= \mathbf{H}_r^{S/S^*} \cdot \hat{\mathbf{y}} \\ \mathcal{J}_{3r}^H &= \mathbf{H}_r^{S/S^*} \cdot \hat{\mathbf{z}} \end{aligned} \quad (3.17)$$

The *partial momenta* can be formulated using the partial momenta of bodies, which in turn can be formulated with the partial velocities in the system. These expressions were derived in section 2.3.

Desired momentum rates (due to control of external forces and torques) can be expressed in terms of these Jacobian augmentation equations and their derivatives along with the generalized speeds and their derivatives.

$$\begin{aligned} \dot{\mathbf{L}}^S &= \mathcal{J}^L \dot{u} + \dot{\mathcal{J}}^L u \\ \dot{\mathbf{H}}^{S/S^*} &= \mathcal{J}^H \dot{u} + \dot{\mathcal{J}}^H u \end{aligned} \quad (3.18)$$

The derivatives of the elements of the augmented Jacobian can be determined from the partial momenta:

$$\begin{aligned} \dot{\mathcal{J}}_{1r}^L &= \dot{\mathbf{L}}_r^S \cdot \hat{\mathbf{x}} \\ \dot{\mathcal{J}}_{2r}^L &= \dot{\mathbf{L}}_r^S \cdot \hat{\mathbf{y}} \\ \dot{\mathcal{J}}_{3r}^L &= \dot{\mathbf{L}}_r^S \cdot \hat{\mathbf{z}} \end{aligned} \quad (3.19)$$

and

$$\begin{aligned}\dot{j}_{1r}^H &= \dot{\mathbf{H}}_r^{S/S^*} \cdot \hat{\mathbf{x}} \\ \dot{j}_{2r}^H &= \dot{\mathbf{H}}_r^{S/S^*} \cdot \hat{\mathbf{x}} \\ \dot{j}_{3r}^H &= \dot{\mathbf{H}}_r^{S/S^*} \cdot \hat{\mathbf{x}}\end{aligned}\quad (3.20)$$

where the derivatives, taken in a Newtonian reference frame, of the partial momenta are:

$$\begin{aligned}\dot{\mathbf{L}}_r^S &\triangleq \frac{d^N}{dt} \mathbf{L}_r^S \\ \dot{\mathbf{H}}_r^{S/S^*} &\triangleq \frac{d^N}{dt} \mathbf{H}_r^{S/S^*}\end{aligned}\quad (3.21)$$

and the rate of change of the momenta are given by:

$$\dot{\mathbf{L}}^S = \sum \mathbf{F}_i^{\text{ext}} \quad (3.22)$$

$$\dot{\mathbf{H}}^{S/S^*} = \sum \mathbf{T}^{\text{ext}} + \sum (\mathbf{r}^{\text{ext}} - \mathbf{r}^*) \times \mathbf{F}^{\text{ext}} \quad (3.23)$$

This completes the formal description of how to augment the manipulator Jacobian with *partial momenta*. Momentum can be included as part of the control objectives of the CT controller by augmenting the Jacobian matrix with the terms described in equations 3.16 and 3.17, and the derivative of the Jacobian matrix with the terms described in equations 3.19 and 3.20. Momentum can be controlled if external forces and torques can be applied, otherwise, momentum conservation (a momentum rate of 0) can be specified explicitly.

3.3.3 Control of Constrained Systems

If a system S has a complete⁵ motion constraint, it will possess 3 fewer degrees of freedom in 3D than when unconstrained. For example, consider a closed-chain mechanism, such as a cooperating-arm robot, which is completely constrained in linear motion (and may have some angular motion constraints as well) at every joint in the chain. Such a chain, if cut, would have more degrees of freedom.

In a dynamic system with such nonholonomic constraints, the generalized speeds $u_{1..n}$ are not independent. Motion constraints define relations that enforce certain relationships

⁵Involving all degrees of freedom in either angular or linear motion

between rates of motion in the system. This analysis will examine a velocity constraint introduced when performing cooperating-arm manipulation for the simple case where no angular-velocity constraints exist. Velocity constraints express the identical motion of coincident endpoints at any arbitrary cut in the chain. This analysis is applicable to more than one simultaneous constraint.

The velocity constraint of chain closure, where a point p , the closure point, on a manipulator is coincident with a point p' is

$$\mathbf{v}^p = \mathbf{v}^{p'} \quad (3.24)$$

In section 2.4.1 a *constraint* velocity was defined as:

$$\begin{aligned} \mathbf{c} &\stackrel{\Delta}{=} \mathbf{v}^p - \mathbf{v}^{p'} \\ &= 0 \end{aligned} \quad (3.25)$$

and the *constraint partial velocities*⁶ evaluated to:

$$\mathbf{c}_r \stackrel{\Delta}{=} \mathbf{v}_r^p - \mathbf{v}_r^{p'}$$

It is evident that by dot multiplication with inertial basis vectors, as was done with endpoint velocity in the previous section, this vector equation can be reduced to scalar equations for incorporation into the system Jacobian.

$$0 \stackrel{\Delta}{=} \mathcal{J}_{3.7}^C u$$

where the elements of these Jacobian augmentation equations are:

$$\begin{aligned} \mathcal{J}_{1r}^C &= \mathbf{c}_r \cdot \hat{\mathbf{x}} \\ \mathcal{J}_{2r}^C &= \mathbf{c}_r \cdot \hat{\mathbf{y}} \\ \mathcal{J}_{3r}^C &\stackrel{\Delta}{=} \mathbf{c}_r \cdot \hat{\mathbf{z}} \end{aligned} \quad (3.26)$$

These constraint partial velocities can be formulated automatically using the partial velocities of the points that are touching.

⁶Although the constraint velocity is zero, the individual *constraint partial velocities* are non-zero.

By differentiating the constraint augmentation equations, the acceleration constraints turn out to be:

$$0 = \mathcal{J}^C \dot{u} + \dot{\mathcal{J}}^C u \quad (3.27)$$

The derivatives of the constraint augmentation equations can also be determined from the partial velocity derivatives:

$$\begin{aligned} \dot{j}_{1r} &= \dot{\mathbf{c}}_r \cdot \mathbf{x} \\ \dot{j}_{2r} &= \dot{\mathbf{c}}_r \cdot \mathbf{y} \\ \dot{j}_{3r} &\stackrel{3.8}{=} \dot{\mathbf{c}}_r \cdot \mathbf{z} \end{aligned} \quad (3.28)$$

where the derivatives, taken in a Newtonian reference frame, of the constraint partial velocities are combinations of endpoint partial velocities:

$$\begin{aligned} \dot{\mathbf{c}}_r &\stackrel{2.36}{=} \frac{d}{dt} \mathbf{c}_r \\ &= \dot{\mathbf{v}}_r^p - \dot{\mathbf{v}}_r^{p'} \end{aligned}$$

Angular velocity constraints can be derived in an identical manner to linear velocity constraints, just substitute ω for \mathbf{v} , and substitute bodies B and B' for points p and p'.

This completes the formal description of how to include dynamic constraints in the control objectives by augmenting the manipulator Jacobian with *partial constraint velocities* described in equation 3.26, and the derivative of the Jacobian with the terms described in equation 3.28. By augmenting the Jacobian with dynamic constraints, a *complete, consistent* set of generalized accelerations will be determined when the Jacobian equation is solved.

3.4 Resolving Generalized Accelerations

Suppose we wish to control a two-armed free-flying robot using cooperating arms to manipulate a payload. The control objectives are not only to control the position and orientation of the object, but also the linear and angular momentum of the system. Furthermore, a dynamic constraint (the closed kinematic chain) needs to be accounted for. The control

objectives are:

$$\mathbf{a}^S \triangleq \begin{bmatrix} \mathbf{a}^{\text{endpoint}} \\ \frac{d}{dt} \mathbf{L}^S \\ \frac{d}{dt} \mathbf{H}^{S/S^*} \\ 0 \end{bmatrix} \quad (3.29)$$

An augmented Jacobian \mathcal{J} can now be constructed as follows: the manipulator portion of the Jacobian \mathbf{J} relates payload speeds and body angular rates to the the system's generalized speeds. Augmentation equations describe the system linear and angular momenta: \mathcal{J}^L , \mathcal{J}^H . Finally, augmentation equations that describe dynamic constraints, \mathcal{J}^C , are added if the system has motion constraints. This process results in a full rank⁷ Jacobian that looks like:

$$\mathcal{J}_{3.1,3.29} = \begin{bmatrix} \mathbf{J} \\ \mathcal{J}^L \\ \mathcal{J}^H \\ \mathcal{J}^C \end{bmatrix}_{n \times n}$$

This augmented Jacobian describes the relationship between the generalized speeds and specific quantities in the dynamic system as follows:

$$\mathbf{v}^S = \mathcal{J}^S \mathbf{u} \quad (3.30)$$

$$= \begin{bmatrix} \mathbf{v}^{\text{endpoint}} \\ \mathbf{L}^S \\ \mathbf{H}^{S/S^*} \\ 0 \end{bmatrix} \quad (3.31)$$

These control objectives enter the Jacobian equation:

$$\mathbf{a}^S = \mathcal{J} \dot{\mathbf{u}} + \dot{\mathcal{J}} \mathbf{u} \quad (3.32)$$

and the generalized accelerations corresponding to this set of control objectives can be determined:

$$\dot{\mathbf{u}} = \mathcal{J}^{-1}(-\dot{\mathcal{J}} \mathbf{u} + \mathbf{a}^S) \quad (3.33)$$

⁷Of rank n , where n is the number of degrees of freedom of the system

These generalized accelerations (derivatives of the generalized speeds) can be used in an inverse dynamics routine to calculate manipulator control torques. By augmenting the manipulator Jacobian matrix with momentum and/or dynamic constraint terms, it is possible to use conventional CT solution equations. This shows that standard CT solution techniques are, in fact, applicable to a much larger range of complex dynamic systems.

3.5 Examples of Augmented Jacobian Matrices

In this section several examples are presented of augmented Jacobian matrices for control of complex dynamic systems. Examples include Jacobian matrices applicable to fixed-base or free-flying robots with constrained or unconstrained (closed-chain) dynamics. In all cases, augmenting the manipulator Jacobian makes it full rank, making it invertible (when not singular), allowing the generalized accelerations to be solved for in the computed-torque control problem.

A free-flying robot has two arms whose endpoints are designated by p^1 and p^2 . The objectives are to control the positions of these two endpoints using second-order error control laws. The outputs from the error controllers are desired endpoint accelerations. No external forces or torques are to be applied to the robot in this phase of its task: desired momentum rates are zero. The augmented Jacobian matrix for this 12th order system,

taking into account the system's linear and angular momenta is:

$$\mathbf{J} = \begin{bmatrix} \mathbf{L}_{1x} & \mathbf{L}_{2x} & \dots & \mathbf{L}_{nx} \\ \mathbf{L}_{1y} & \mathbf{L}_{2y} & & \mathbf{L}_{ny} \\ \mathbf{L}_{1z} & \mathbf{L}_{2z} & & \mathbf{L}_{nz} \\ \mathbf{H}_{1x} & \mathbf{H}_{2x} & & \mathbf{H}_{nx} \\ \mathbf{H}_{1y} & \mathbf{H}_{2y} & & \mathbf{H}_{ny} \\ \mathbf{H}_{1z} & \mathbf{H}_{2z} & & \mathbf{H}_{nz} \\ \mathbf{v}_{1x}^{p^1} & \mathbf{v}_{2x}^{p^1} & & \mathbf{v}_{nx}^{p^1} \\ \mathbf{v}_{1y}^{p^1} & \mathbf{v}_{2y}^{p^1} & & \mathbf{v}_{ny}^{p^1} \\ \mathbf{v}_{1z}^{p^1} & \mathbf{v}_{2z}^{p^1} & & \mathbf{v}_{nz}^{p^1} \\ \mathbf{v}_{1x}^{p^2} & \mathbf{v}_{2x}^{p^2} & & \mathbf{v}_{nx}^{p^2} \\ \mathbf{v}_{1y}^{p^2} & \mathbf{v}_{2y}^{p^2} & & \mathbf{v}_{ny}^{p^2} \\ \mathbf{v}_{1z}^{p^2} & \mathbf{v}_{2z}^{p^2} & & \mathbf{v}_{nz}^{p^2} \end{bmatrix}_{12 \times 12} \quad (3.34)$$

In order to simplify presentation, a short-form notation will be used, where the inclusion of a vector, such as \mathbf{H}_1 in a matrix implies that all its elements $(\mathbf{H}_1 \cdot \hat{\mathbf{x}}) \dots$ are present as a column. The Jacobian matrix in equation 3.34 is then

$$\mathbf{J} = \begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_2 & \dots & \mathbf{L}_{12} \\ \mathbf{H}_1 & \mathbf{H}_2 & & \mathbf{H}_{12} \\ \mathbf{v}_1^{p^1} & \mathbf{v}_2^{p^1} & & \mathbf{v}_{12}^{p^1} \\ \mathbf{v}_1^{p^2} & \mathbf{v}_2^{p^2} & & \mathbf{v}_{12}^{p^2} \end{bmatrix}_{12 \times 12} \quad (3.35)$$

A fixed-base two-armed robot is to use two arms to grasp an object and manipulate it. The objectives are to control the orientation of the object B , and the position of a point B_0 on it. Error controllers provide desired accelerations for these quantities. The augmented Jacobian for this system to control the acceleration of the point B_0 , the angular acceleration of body B , and the constraint at point p (at one of the joints, selected by the analyst), is:

$$\mathbf{J} = \begin{bmatrix} \mathbf{v}_1^{B_0} & \mathbf{v}_2^{B_0} & \dots & \mathbf{v}_n^{B_0} \\ \boldsymbol{\omega}_1^B & \boldsymbol{\omega}_2^B & & \boldsymbol{\omega}_n^B \\ \mathbf{c}_1^p & \mathbf{c}_2^p & & \mathbf{c}_n^p \end{bmatrix}_{n \times n} \quad (3.36)$$

The inverse dynamics can accept additional information about the internal force at the point p in order to allow control of the object squeeze force. An example of a free-flying closed-chain system is presented next.

A free-flying robot is to use two arms to grasp an object and manipulate it. The objectives are to control a point B^* of the object B , the orientation of the object, and system linear and angular momentum. Error controllers provide desired momentum rates, desired accelerations of the point B^* and desired angular accelerations of body B . The augmented Jacobian for this system, taking into account the system's linear and angular momenta and the constraint at point p (a virtual cut at one of the joints, selected by the analyst) is:

$$\mathbf{J} = \begin{bmatrix} \mathbf{L}_1 & \mathbf{L}_2 & \dots & \mathbf{L}_n \\ \mathbf{H}_1 & \mathbf{H}_2 & & \mathbf{H}_n \\ \mathbf{v}_1^{B^*} & \mathbf{v}_2^{B^*} & & \mathbf{v}_n^{B^*} \\ \boldsymbol{\omega}_1^{B^*} & \boldsymbol{\omega}_2^{B^*} & & \boldsymbol{\omega}_n^{B^*} \\ \mathbf{c}_1^p & \mathbf{c}_2^p & & \mathbf{c}_n^p \end{bmatrix}_{n \times n} \quad (3.37)$$

As in the previous example, it is possible to control the object squeeze force by specifying a desired internal force at the point p .

3.5.1 Summary of Examples

The above examples show that the structure of the Jacobian is very uniform when expressed using partial velocities, *partial momenta* and *partial constraint velocities*. This knowledge of structure can be used to formulate the entries of the Jacobian matrix automatically, Chapter 4 discusses a computer program that does this. For example, consider the Jacobian entries for the velocity of a point: the entries for the Jacobian are the components of the partial velocities expressed along the inertial \hat{x} , \hat{y} and \hat{z} axes. If these partial velocities are known, then no additional work need be done in order to determine the Jacobian entries, their values can be copied directly into the corresponding row(s) of the Jacobian.

3.6 Inverse Dynamics

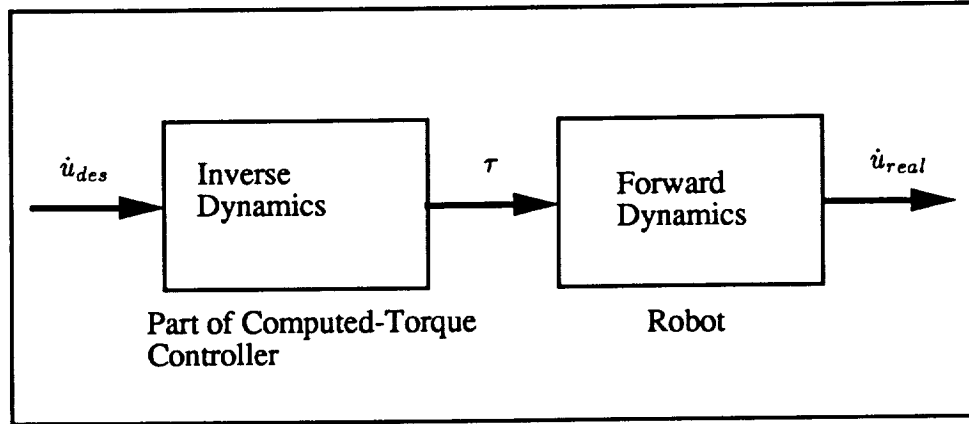


Figure 3.2: **Inverse and Forward Dynamics**

With an accurate dynamics model in the computed-torque controller, the accelerations in the dynamic system can accurately match those desired.

Inverse dynamics is the process of attempting to ‘invert’ the plant dynamics so that the desired generalized accelerations, determined via the Jacobian transformation from the error controllers, can be achieved by commanding joint torques to the manipulator⁸. Inverse dynamics requires accurate knowledge of the system’s geometric and mass properties, as well as knowledge of the full system state. If significant, accurate modeling of friction, flexibility, and other properties of the system that affect the dynamics, will be required. Inverse dynamics for both unconstrained and constrained rigid-body systems will be discussed here.

3.6.1 Unconstrained System Inverse Dynamics

Rigid multibody inverse dynamics can be solved using the generalized dynamics matrix equation derived in section 2.3: generalized forces, and hence manipulator torques and forces, can be determined given the desired system generalized accelerations.

$$\mathbf{F} = \mathbf{M} \dot{\mathbf{u}}_{des} + \mathbf{N} \mathbf{u}$$

$$\boldsymbol{\tau} = \mathbf{W}^{-1} \mathbf{F}$$

⁸Computed-torques could be fed to tight torque-control loops wrapped around the manipulator joints in the event that joints are not friction-free.

where the matrix \mathbf{W} maps generalized speeds into the derivatives of the generalized coordinates (joint rates).

$$\dot{\mathbf{q}}_r \stackrel{2.5}{=} \sum_{s=1}^n \mathbf{W}_{rs} u_s$$

As indicated previously, the inverse of the matrix \mathbf{W} is typically trivial and time-invariant, and depends on the analyst's choice of generalized speeds.

While manipulator torques can be solved using this equation, it is not numerically efficient: the matrices have n^2 entries, and require re-evaluation at every time step. A more efficient method for performing inverse dynamics using a recursive algorithm will be presented in the next chapter, which discusses recursive dynamics as applicable to various aspects of computed-torque control and dynamics simulation.

3.6.2 Constrained-System Inverse Dynamics: The Hard Way

The dynamics equations get more complex for constrained systems, as was shown in section 2.4.4. A reduced-order set of matrix of equations that takes into account the motion constraints is even more computationally expensive to formulate than the unconstrained equations described above. First, one needs to determine the constraining matrix:

$$\mathbf{A} \stackrel{2.51}{=} -(\mathbf{C}^c)^{-1} \mathbf{C}^u$$

and then one can determine the reduced-order equations of motion.

$$\begin{aligned} \tilde{\mathbf{M}} &\stackrel{2.54}{=} \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{M} \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix} \\ \tilde{\mathbf{N}} &\stackrel{2.54}{=} \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{N} \\ \tilde{\mathbf{W}} &\stackrel{2.54}{=} \begin{bmatrix} \mathbf{I} \\ \mathbf{A} \end{bmatrix}^T \mathbf{W} \end{aligned}$$

From the reduced-order equations of motion the nonholonomic system's generalized forces $\tilde{\mathbf{F}}$ can be determined as follows:

$$\tilde{\mathbf{F}}_{1..p} = \tilde{\mathbf{M}} \dot{u}_{1..p} + \tilde{\mathbf{N}} u_{1..n} \quad (3.38)$$

$$\tilde{\mathbf{W}}_{p \times n} \tau_{n \times 1} = \tilde{\mathbf{F}}_{p \times 1} \quad (3.39)$$

Joint torques, however, cannot be solved for because the matrix $\tilde{\mathbf{W}}$ is not full rank. It is necessary to introduce additional equations, for example extra equations of motion involving internal forces, to resolve the problem. These additional equations can allow a constraint force (such as an object squeeze force) to be specified. This is a complicated process: an alternate, simpler method is presented instead.

3.6.3 Constrained-System Inverse Dynamics: An Easier Way

It turns out that if the augmented Jacobian is used to determine generalized accelerations for the system, then *unconstrained* dynamic equations can be used to perform inverse dynamics. This is because the constraint equations embedded in the constrained dynamics have been taken into account when formulating the augmented Jacobian; the generalized accelerations provided to the inverse dynamics are *already* consistent with the motion constraints.

$$\mathcal{J} \stackrel{3.1}{=} \begin{bmatrix} \mathbf{J} \\ \mathcal{J}^L \\ \mathcal{J}^H \\ \mathcal{J}^C \end{bmatrix}$$

$$\dot{\mathbf{u}} \stackrel{3.33}{=} \mathcal{J}^{-1}(-\dot{\mathcal{J}}\mathbf{u} + \mathbf{a}^S)$$

This simple, powerful, step allows unconstrained inverse dynamics to be applied to any constrained system that can be modeled with the augmented Jacobian technique: a category that includes all closed-chain manipulators. The unconstrained dynamics can include forces at the endpoints, so that desired internal forces in the manipulator (squeeze force) can be specified. Unconstrained inverse dynamics can be solved easily with a recursive Newton-Euler algorithm, to be presented in the next chapter.

3.7 Summary

This chapter introduced and examined two key components of the computed-torque (CT) control technique: the Jacobian equation, and inverse dynamics. The manipulator Jacobian

was augmented in order to include linear and angular momentum and dynamic constraints in the control objectives of free-flying robots.

Inverse dynamics techniques using equations of motion for both constrained and unconstrained systems were presented. When using the augmented Jacobian to solve for system generalized accelerations, it is possible to use unconstrained inverse dynamics – even for systems with dynamic constraints. The simplified techniques applicable to solving unconstrained dynamic systems, such as the recursive technique discussed in the next chapter, then extend to *all* constrained systems whose constraints can be expressed as discussed in section 3.3.3: velocity and/or angular velocity constraints.

Chapter 4

Recursive Dynamics

Most robotic manipulators consist of links connected to one another in a serial manner, offering many degrees of freedom. This mechanical arrangement, a kinematic chain, has special properties for dynamic modeling. In this chapter the generalized dynamics expressions derived in chapters 2 and 3 are specialized to this class of robots, kinematic chains, with recursive relations. Kinematics and dynamics expressions for kinematic chains can be formulated in terms of local effects and effects due to previous bodies in the chain. This is applicable to a variety of multi-link robotic manipulators: fixed-base robots, free-flying robots, multi-armed robots, and robots with cooperating arms.

Three key components of computed-torque control are constructed from recursively evaluated components. These recursive relations are further specialized to two dimensions and used to implement an automated computed-torque control computer program (RD). This program is used in following chapters for simulation and for experimental control of complex dynamic systems.

4.1 The Kinematic Chain

A kinematic chain is a set of serially connected bodies, where joints between bodies are revolute and/or prismatic. Bodies are typically connected in a serial fashion such as a SCARA robot, shown in Figure 4.1. Chains may also have tree-like branches - multiple manipulator arms - such as SPAR Aerospace's Special Purpose Dextrous Manipulator

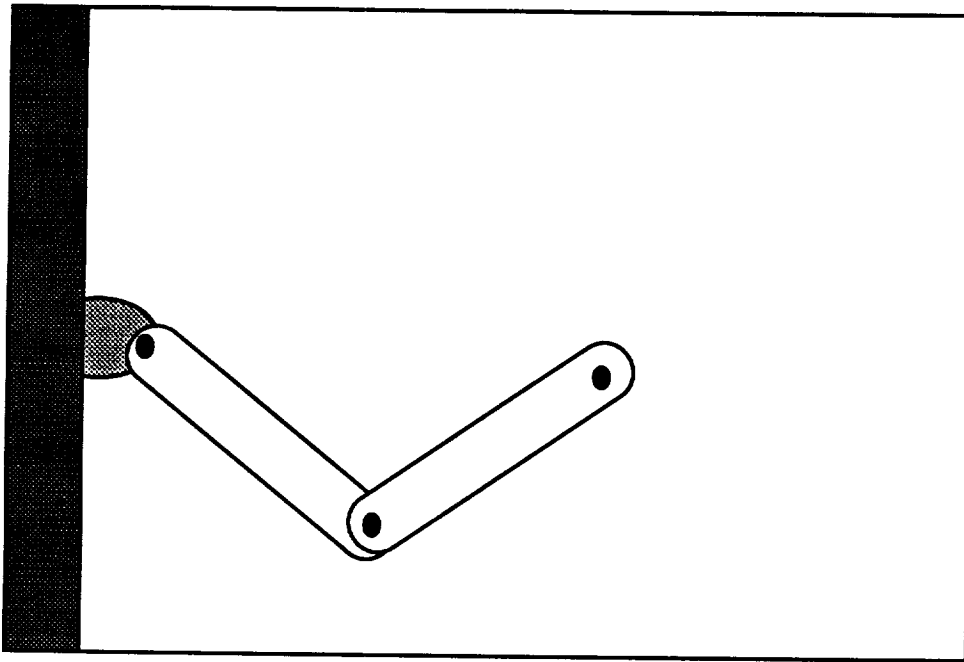


Figure 4.1: **A Standard Configuration Assembly Robot Arm**

The SCARA robot, a widely used type of manipulator, is a simple kinematic chain.

(SPDM). Kinematic chains can be fixed-base, such as the robot shown in Figure 4.1, or they can be free-flying, such as NASA's proposed Orbital Maneuvering Vehicle (OMV).

First, the recursive structure of kinematics and dynamics terms in kinematic chains will be examined. This is followed by optimizations for the two-dimensional case, so that the recursive relations can be efficiently implemented in a computer program. A summary of the capabilities of the **RD** computer program is presented.

4.2 Computed-Torque Control on Kinematic Chains

In chapter 3, the seven components of a computed-torque controller were shown in figure 3.1 on page 30. Of these seven, three key components: the Jacobian, the inverse dynamics, and the sensors (a mapping between states and outputs) form the essence of the linearizing, decoupling computed-torque controller. To recap these three components:

- The Jacobian equation translates control objectives into generalized, possibly joint, accelerations. The Jacobian matrix needs to be inverted to accomplish this.

- The inverse dynamics section uses these computed generalized accelerations to calculate the required manipulator joint torques using knowledge of the dynamic system. Compensation torques for nonlinear effects are computed using measurements of the state: joint positions and angular rates. It is assumed that the computed torques can be delivered to the links by the actuators directly, or via a tight torque control loop wrapped around the actuators. If the dynamic model is accurate, these torques will achieve closely the desired control objectives.
- The sensors provide measurements of quantities of interest, and can be used by the feedback controllers to achieve desired system response. These signals are typically functions of the state, such as endpoint position and velocity, rather than a part of the state.

It has already been shown in chapters 2 and 3 that general kinematics, dynamics and computed-torque control expressions can be formulated using partial velocities. It turns out that partial velocities can be easily evaluated in kinematic chains using recursive techniques. The analyst can choose the system generalized speeds to achieve computation optimizations.

There is a minimal set of quantities that need to be computed for computed-torque control using this method. For dynamic terms and/or momentum terms, the partial velocities and their derivatives of mass center points need to be computed. For Jacobian entries, the partial velocities and their derivatives of controlled endpoints (or joints, if under joint control) need to be computed. For simulation signal outputs, the partial velocities and their derivatives for points of interest need to be computed.

4.3 Recursive Formulation of Kinematic Terms

Recursion:

The determination of a succession of elements (as numbers or functions) by operation on one or more preceding elements according to a rule or formula involving a finite number of steps.

The kinematic chain is ideally suited to recursive techniques. An elementary examination shows that an endpoint's position and velocity depend on the positioning and motions of all preceding links. This is applicable to more than just continuous single chains: branches in the chains are possible (such as with a multi-armed robot). Given a topology of a dynamic system, the positions and velocities of points on bodies can be expressed recursively, in terms of the positions and velocities of previous points on bodies along a chain.

Recursive solutions have been shown to be computationally more efficient than matrix-oriented solutions by Nielan [21] and Wampler [39], and commercial symbolic dynamics codes, such as SDEXACT [27] use recursion to formulate their solutions. Recursive techniques are tied to the chain structure that they operate on, the number of operations is *typically* on the order of the number of bodies, n , not the number of elements in a matrix n^2 . Matrix elements, needed in simulation equations, can be formulated from recursive terms when needed. Recursive algorithms for determining kinematic terms, where joints between bodies are revolute, are presented in this section.

4.3.1 Positions of Points

A recursive relation that describes the position of a point \mathbf{p}_i^j on a link i is

$$\mathbf{p}_i^j = \mathbf{p}_i^0 + \mathbf{r}^j \quad (4.1)$$

where \mathbf{r}^j is the distance from the base point of the link located on the axis $\hat{\lambda}_i$ (as defined by the Denavit-Hartenberg convention, or any other) to the point of interest.

4.3.2 Velocities of Points

Recursive relations for describing the velocity of a point \mathbf{p}_i^j on a link i , with respect to a previous link $i - 1$, are:

$$\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} + \dot{q}_i \hat{\lambda}_i \quad (4.2)$$

$$\mathbf{v}^j = \mathbf{v}^{\mathbf{p}_i^j} + \boldsymbol{\omega}^i \times \mathbf{r}^i \quad (4.3)$$

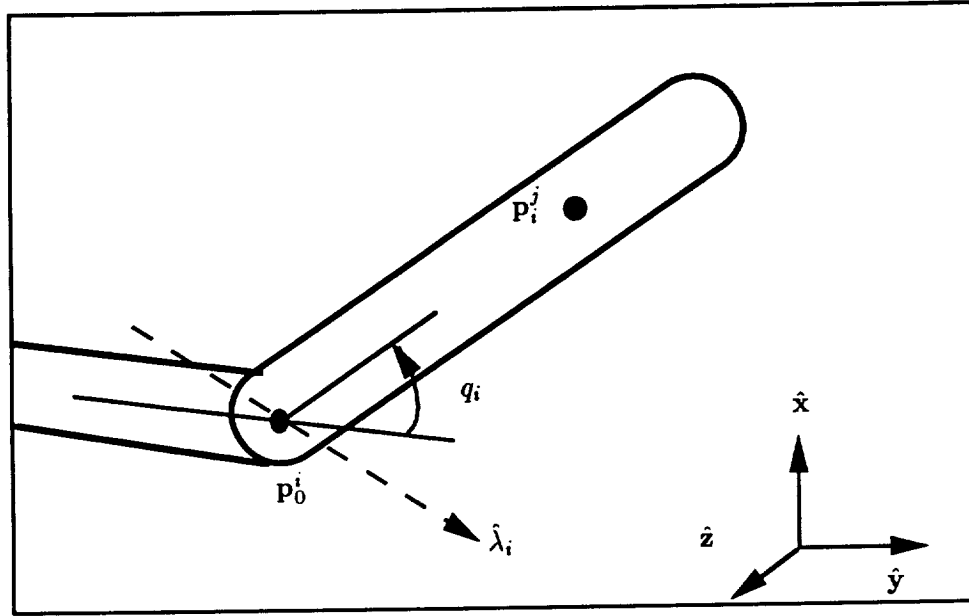


Figure 4.2:

A Single Link in a Serial Chain

Velocities of points and angular velocities for any link can be expressed in terms of conditions at a preceding body, knowing the connection point P_0^i , the axis $\hat{\lambda}_i$, and the joint angular speed.

4.3.3 Partial Velocities

Velocity relations can be broken down into their partial velocity components. Partial velocities are used in this thesis to construct the kinematic and dynamic terms used in computed-torque control and dynamic simulation. Their recursion relations are:

$$\omega_r^i = \omega_r^{i-1} + \hat{\lambda}_i \sum_{s=1}^n W_{rs} u_s \quad (4.4)$$

$$v_r^{P_j^i} = v_r^{P_0^i} + \omega_r^i \times r^i \quad (4.5)$$

where the matrix W is the map between generalized coordinate rates and generalized speeds. It is dependent on the analyst's *definition* of the generalized speeds. The choice of generalized speeds is arbitrary, but must ensure that the matrix W is invertible. A common choice of generalized speeds for 3D systems and a good choice of generalized speeds for 2D systems, that yields a very simple formulation, are discussed next.

4.3.4 Choice of Generalized Speeds

The partial angular velocities, used to determine partial velocities along a chain, are very dependent on the definitions of the generalized speeds. The generalized speeds and the coordinate rates are linearly related via the equation

$$\dot{q}_r \stackrel{2.27}{=} \sum_{s=1}^n \mathbf{W}_{rs} u_s$$

In 3D systems, the generalized speeds are typically chosen to be the derivatives of the generalized coordinates: $u_r \triangleq \dot{q}_r$. The map \mathbf{W} is then:

$$\mathbf{W} = \mathbf{I}_{n \times n} \quad (4.6)$$

and the relation between the partial angular velocities along the chain (from start to end) is:

$$\omega_r^i = \begin{cases} \omega_r^i & \text{for } r < i \\ \hat{\lambda}_i & \text{for } r = i \\ 0 & \text{for } r > i \end{cases} \quad (4.7)$$

In 2D systems, a good choice for the generalized speeds are the measures of the angular velocities about the $\hat{\mathbf{z}}$ axis: $u_r \triangleq \omega^i \cdot \hat{\mathbf{z}}$. This choice results in fewer calculations to evaluate terms in the kinematics and dynamics expressions. This is particularly evident in the simplified form of the partial angular velocity terms, equation 4.9, and partial velocity terms, equation 4.31: most evaluate to zero, or are constant over a set. The $\hat{\mathbf{z}}$ axis is perpendicular to the plane of motion.

The map between generalized coordinate rates and generalized speeds is then:

$$\mathbf{W} = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ -1 & 1 & 0 & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots \\ 0 & 0 & -1 & 1 & \text{etc.} \end{bmatrix}_{n \times n} \quad (4.8)$$

and the relation between the partial angular velocities along the chain (from start to end) is:

$$\omega_r^i = \begin{cases} \hat{\mathbf{z}} & \text{for } r = i \\ 0 & \text{for } r \neq i \end{cases} \quad (4.9)$$

Since most of the partial angular velocities are zero, this choice of generalized speeds can be used to simplify greatly and to speed up the formulation (and computation) of kinematic terms. These optimizations, possible with 2D systems, are explored further in section 4.5.

4.3.5 Accelerations of Points

The recursive relation for describing the angular acceleration of body α^i with respect to the angular acceleration of the previous link α^{i-1} is:

$$\alpha^i = \alpha^{i-1} + \ddot{q}_i \hat{\lambda}_i + \dot{q}_i \omega^i \times \hat{\lambda}_i \quad (4.10)$$

$$(4.11)$$

The recursive relation for describing the acceleration of a point $\mathbf{a}^{P_j^i}$ on a link i , with respect to the acceleration of a point on the joint with the previous link $\mathbf{a}^{P_j^0}$, is:

$$\begin{aligned} \mathbf{a}^{P_j^i} &= \mathbf{a}^{P_j^0} + \mathbf{a}^{P_j^{i/0}} \\ &= \mathbf{a}^{P_j^i} + \alpha^i \times \mathbf{r}^i + \omega^i \times \omega^i \times \mathbf{r}^i \end{aligned} \quad (4.12)$$

It was shown in section 2.2.2 that accelerations could be factored into terms involving the partial velocities, the generalized accelerations, the derivatives of the partial velocities, and the generalized speeds:

$$\begin{aligned} \mathbf{a}^i &= \sum_{r=1}^n \mathbf{v}_r^i \dot{u}_r + \sum_{r=1}^n \dot{\mathbf{v}}_r^i u_r \\ \alpha^i &= \sum_{r=1}^n \omega_r^i \dot{u}_r + \sum_{r=1}^n \dot{\omega}_r^i u_r \end{aligned}$$

The recursive formulation of partial velocities has already been discussed, the formulation of the derivative of the partial velocities is discussed next.

4.3.6 Derivatives of Partial Velocities

The derivatives of partial velocities can be formulated recursively using the following relation

$$\dot{\omega}_r^i = \dot{\omega}_r^{i-1} + \hat{\lambda}_i \sum_{s=1}^n \mathbf{W}_{rs} \dot{u}_s + \omega^i \times \hat{\lambda}_i \sum_{s=1}^n \mathbf{W}_{rs} u_s \quad (4.13)$$

$$\dot{\mathbf{v}}_r^{P_j^i} = \dot{\mathbf{v}}_r^{P_j^i} + \dot{\omega}_r^i \times \mathbf{r}^i \quad (4.14)$$

4.4 Application to Computed-Torque Control

4.4.1 Jacobian Equation Elements

The Jacobian equation, used to solve for desired system generalized accelerations given control objectives, is:

$$\dot{\mathbf{u}} = \mathcal{J}_{3.33}^{-1}(-\dot{\mathcal{J}}\mathbf{u} + \mathbf{a}^S)$$

The augmented Jacobian can be used to express endpoint velocities, body angular rates, linear and angular momentum, and dynamic constraints in terms of the generalized speeds. The discussion in sections 3.2 and 3.3 showed how to formulate Jacobian matrix entries on a row-by-row basis, according to the control objectives.

The rows of the Jacobian matrix are the partial velocities and/or the partial momenta. So once these are determined they can be used as-is in the Jacobian.

4.4.2 Recursive Inverse Dynamics

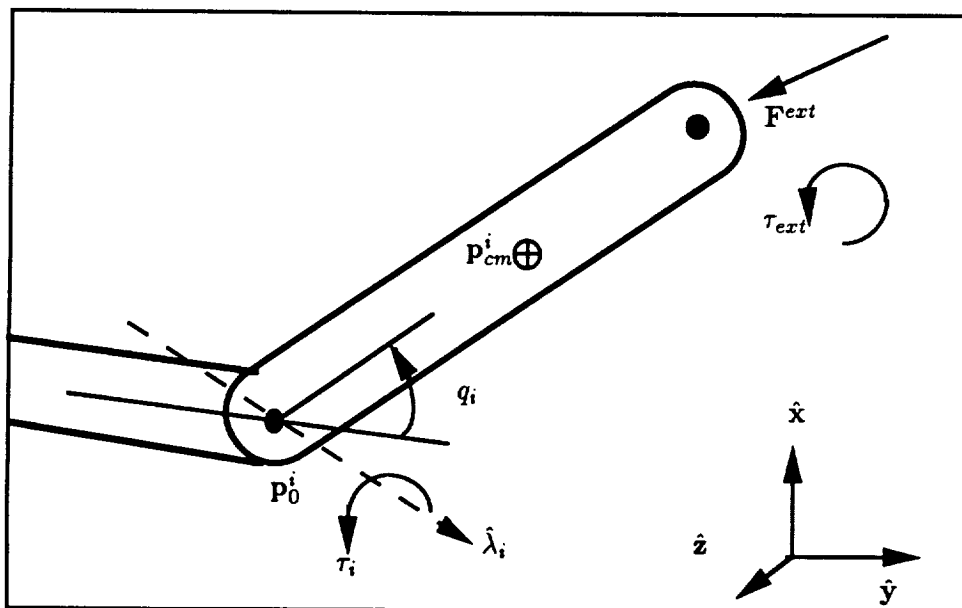


Figure 4.3: Inverse Dynamics on a Link-by-Link Basis

Many computed torque control schemes use the following matrix equation to compute the joint torques:

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{V}(\mathbf{q}, \dot{\mathbf{q}}) + \boldsymbol{\tau}$$

Nielan [21] showed that $\mathcal{O}(n^2)$ computations are required to solve this equation. The mass matrix and nonlinear terms of the system need to be evaluated in order to solve for the control torques. Using these equations of motion to solve the inverse dynamics equation is numerically inefficient.

Robot forward dynamics have been solved by Rosenthal [26], Rodriguez [25] and others in $\mathcal{O}(n)$ computations. Luh, Walker and Paul [18] solved inverse dynamics equations for control torques in $\mathcal{O}(n)$ computations using a recursive Newton-Euler algorithm. In this section a similar straightforward algorithm is used to solve the inverse dynamics for the actuator torques along a serial chain. This algorithm consists of two phases: Solving for system accelerations on an outward pass, and solving for forces and torques on an inward pass.

Outward Recursion

In the first part of the algorithm, the accelerations are propagated out from the base of the robot to the end(s) of the kinematic chain(s). Accelerations at the base of the robot are known: either they are zero, or, in the event that the robot is free-flying, they are the derivatives of the generalized speeds for its linear motion. Other generalized speeds can be used to describe the free-flying robot body's angular motion. A free-flying robot base is effectively the first link in the chain.

The joint accelerations (derived from system generalized accelerations) are used to determine the accelerations of points and angular accelerations of successive bodies in the system. The accelerations of all the relevant points on a body, including that of the center of mass points, will need to be evaluated for each body in the chain. All other accelerations in the system can be determined using the base accelerations and the link recursion relations. The link recursion relation for acceleration at the end of a link states it is related to the acceleration at the start of a rigid body link as follows:

$$\mathbf{a}^{\text{end}} = \mathbf{a}^{\text{start}} + \boldsymbol{\alpha}^{\text{link}} \times \mathbf{r}^{\text{start to end}} + \boldsymbol{\omega}^{\text{link}} \times \boldsymbol{\omega}^{\text{link}} \times \mathbf{r}^{\text{start to end}} \quad (4.15)$$

Accelerations of center-of-mass points of the link need to be computed in order to evaluate the d'Alembert forces on the backwards sweep of the recursion:

$$\mathbf{a}^{i*} = \mathbf{a}^{\text{start}} + \boldsymbol{\alpha}^{\text{link}} \times \mathbf{r}^{i*} + \boldsymbol{\omega}^{\text{link}} \times \boldsymbol{\omega}^{\text{link}} \times \mathbf{r}^{i*} \quad (4.16)$$

where the following components are derived as follows:

$$\boldsymbol{\alpha}^i \stackrel{4.10}{=} \boldsymbol{\alpha}^{i-1} + \ddot{q}_i \hat{\lambda}_i + \dot{q}_i \boldsymbol{\omega}^i \times \hat{\lambda}_i$$

where the axis direction $\hat{\lambda}_i$ is a positive rotation, in a right handed sense, for q_i . The link angular velocity $\boldsymbol{\omega}^i$ and its angular acceleration $\boldsymbol{\alpha}^i$ are updated at each step along the chain. For example, in 3D systems where the generalized speeds are defined as $u_r \triangleq \dot{q}_r$, the update will be

$$\boldsymbol{\omega}^i = \boldsymbol{\omega}^{i-1} + \hat{\lambda}_i u_i \quad (4.17)$$

$$\boldsymbol{\alpha}^i = \boldsymbol{\alpha}^{i-1} + \hat{\lambda}_i \dot{u}_i + \boldsymbol{\omega}^i \times \hat{\lambda}_i u_i \quad (4.18)$$

whereas, for 2D systems where the generalized speeds are defined as $u_r \triangleq \boldsymbol{\omega}^i \cdot \hat{\mathbf{z}}$, the update will be

$$\boldsymbol{\omega}^i = \hat{\lambda}_i u_i \quad (4.19)$$

$$\boldsymbol{\alpha}^i = \hat{\lambda}_i \dot{u}_i \quad (4.20)$$

Repeat this process, computing link angular velocity and acceleration, center of mass accelerations and link end accelerations, until these accelerations have been computed all the way out to the end(s) of the chain(s).

Inward Recursion

The second part of the algorithm propagates the forces and moments back from the end(s) of the chain(s), computing joint torques along the way. Forces and moments at the end of the chains are known: when the arm when is not in contact with anything, they are zero, when the arm is in contact with another part of the dynamic system, this force is an internal force, calculated from knowledge of the masses, inertias and accelerations of the body with which it is in contact, as well as an extra degree of freedom in force – a

squeeze force¹. Closed kinematic chains have such an internal force (a squeeze force), on every body in the chain. When dealing with constrained dynamic systems, the algorithm traverses each of the chains up to the virtual cuts imposed by the analyst.

Start by taking moments about the joint at the start of the link, and attribute the component along the joint's axis $\hat{\lambda}_i$ to that joint's actuator. The moments due to the center of mass acceleration and the link's angular acceleration are easily evaluated given its mass properties.

$$\mathbf{T}^i = \mathbf{T}^{\text{link end}} + \mathbf{r}^{\text{start to end}} \times \mathbf{F}^{\text{end}} - \mathbf{r}^{\text{start to }*} \times m^i \mathbf{a}^{i*} \quad (4.21)$$

The joint motor torque can then be evaluated.

$$\tau_i = -\mathbf{T}^i \cdot \hat{\lambda}_i \quad (4.22)$$

The moments on the inboard body (to which this link is connected) are then:

$$\mathbf{T}^{i-1} = \mathbf{T}^i - 2\tau_i \hat{\lambda}_i \quad (4.23)$$

and likewise, the sum of the forces exerted on the inboard body at the connection point are:

$$\mathbf{F}^{\text{end}_{i-1}} = \sum \mathbf{F}^{\text{end}_i} - m^i \mathbf{a}^{i*} \quad (4.24)$$

The focus can now be shifted to the next link in, where this process can be repeated until all of the control torques have been determined. If linear actuators are being used, then the actuator force solution can be done using the sum of forces along the actuator axis.

This Newton-Euler algorithm for solving for the joint control torques (or forces) is straightforward and easily implemented. If the robot has two or more arms, the solution for the joint torques for the various arms can be done in parallel.

4.5 Recursive Dynamics (RD) in Two Dimensions

Several opportunities to optimize computations arise when considering two-dimensional dynamic systems. Combined with the structure inherent in kinematic chains, this results

¹The squeeze force is equal to the subtraction of two forces exerted on a body: it does no work.

in some significant simplifications in the formulations of terms. The derivation of kinematics and dynamics presented here first deals with the simple case of a single fixed-base kinematic chain. After this, the modifications applicable to free-flying systems and more complex branching chain structures are discussed.

4.5.1 Kinematics for Planar Serial Link Manipulators

In particular, all the angular velocities are along the \hat{z} axis, which allows the definition of the generalized speeds in a manner that makes the equations of motion easier to express:

$$u_i \triangleq \omega^i \cdot \hat{z} \quad (4.25)$$

This definition allows angular velocities to be expressed as a function of only one generalized speed: and facilitates the derivations of many terms.

In a system consisting of a set of i planar serially connected rigid links has a simple relationship relating link endpoint velocities to link basepoint velocities:

$$\mathbf{v}^{\text{endpoint}} = \mathbf{v}^{\text{basepoint}} + \omega^{\text{link}} \times \mathbf{r}^{\text{basepoint to endpoint}} \quad (4.26)$$

if this is expressed using partial velocities,

$$\mathbf{v}_r^{\text{endpoint}} = \mathbf{v}_r^{\text{basepoint}} + \omega_r^{\text{link}} \times \mathbf{r}^{\text{basepoint to endpoint}} \quad (4.27)$$

If we define unit vector \hat{x}_i to be along the link, from basepoint to endpoint, and unit vector \hat{y}_i to be perpendicular to \hat{x}_i and in the plane of the manipulator, then we can define \hat{z} , a unit vector perpendicular to the plane of the manipulator as:

$$\hat{z} \triangleq \hat{x}_i \times \hat{y}_i \quad (4.28)$$

The generalized speeds $1..n$ and the angular velocities are related as follows:

$$\omega^{\text{link } i} \stackrel{4.25}{=} u_i \hat{z} \quad (4.29)$$

and the endpoint to basepoint velocity relation for link i of length l_i becomes:

$$\mathbf{r}^{\text{start to end } i} = l_i \hat{x}_i \quad (4.30)$$

and the partial velocity relations along the chain from start $i = 1$ to end n , are as follows:

$$\begin{aligned} \omega_r^i &= \begin{cases} \hat{z} & \text{for } i = r \\ 0 & \text{for } i \neq r \end{cases} \\ \mathbf{v}_r^{\text{endi}} &= \begin{cases} \mathbf{v}_r^{\text{start } i} & \text{for } i = 1 \dots r-1 \\ l_i \hat{\mathbf{y}}_i & \text{for } i = r \\ 0 & \text{for } i = r+1 \dots n \end{cases} \end{aligned} \quad (4.31)$$

The partial velocities of the mass center of each link are calculated using:

$$\mathbf{r}^{\text{start to end } i} = l_i^* \mathbf{x}_i \quad (4.32)$$

$$\hat{z} = \hat{\mathbf{x}}_i^* \times \hat{\mathbf{y}}_i^* \quad (4.33)$$

$$\mathbf{v}^* = \mathbf{v}^{\text{basepoint}} + \omega^{\text{link}} \times \mathbf{r}^{\text{basepoint to endpoint}} \quad (4.34)$$

then, if expressed using partial velocities,

$$\mathbf{v}_r^* = \mathbf{v}_r^{\text{basepoint}} + \omega_r^{\text{link}} \times \mathbf{r}^{\text{basepoint to endpoint}} \quad (4.35)$$

$$\mathbf{v}_r^{i*} = \begin{cases} \mathbf{v}_r^{\text{start } i} & \text{for } i = 1 \dots r-1 \\ l_i \hat{\mathbf{y}}_i & \text{for } i = r \\ 0 & \text{for } i = r+1 \dots n \end{cases} \quad (4.36)$$

These partial velocities can be used to formulate the Jacobian matrix equation, and can also be used for the recursive Newton-Euler inverse dynamics, as discussed earlier in this chapter.

4.5.2 Equations of Motion for Planar Serial Link Manipulators

The formulation of the equations of motion (mass matrix and nonlinear terms) for a kinematic chain with no branches: a highly simplifiable system, are presented in this subsection to illustrate the technique. The kinematic chain has ν bodies, and n degrees of freedom. When no motion constraints are present, the number of degrees of freedom is equal to the number of articulated bodies in the chain. The contribution of body i to the inertia scalar m_{rs} , an element of the mass matrix \mathbf{M} , can be determined as follows:

$$(m_{rs})^i = m^i \mathbf{v}_r^{i*} \cdot \mathbf{v}_s^{i*} + \omega_r^{i*} \cdot \mathbf{I}^{i*} \cdot \omega_s^{i*} \quad (4.37)$$

for the planar system, this reduces to

$$(m_{rs})^i = \begin{cases} \sum_{\text{bodies } i} m^i \mathbf{v}_r^{i*} \cdot \mathbf{v}_s^{i*} & \text{for } s \neq r \\ \sum_{\text{bodies } i} m^i \mathbf{v}_n^{i*} \cdot \mathbf{v}_n^{i*} + \mathbf{I}_{zz}^{i/i*} & \text{for } n = r = s \end{cases} \quad (4.38)$$

where

$$\mathbf{I}_{zz}^{i/i*} = \hat{\mathbf{z}} \cdot \mathbf{I}^{i/i*} \cdot \hat{\mathbf{z}} \quad (4.39)$$

and

$$\omega^{\text{link } i} = u_i \hat{\mathbf{z}} \quad (4.40)$$

A complete inertia scalar can be built from the sum of the effects of all bodies:

$$m_{rs} = m^1 \mathbf{v}_r^{1*} \cdot \mathbf{v}_s^{1*} + m^2 \mathbf{v}_r^{2*} \cdot \mathbf{v}_s^{2*} + \dots \quad (4.41)$$

case $r = s = j$

$$\begin{aligned} m_{jj} &= m^j \mathbf{v}_j^{j*} \cdot \mathbf{v}_j^{j*} + m^j \mathbf{v}_j^{(j+1)*} \cdot \mathbf{v}_j^{(j+1)*} + \dots \\ &= m^j \left(l^{j*} \right)^2 + m^{j+1} \mathbf{v}_j^{\text{end } j} \cdot \mathbf{v}_j^{\text{end } j} + \dots \\ &= m^j \left(l^{j*} \right)^2 \\ &\quad + \left(m^{j+1} + m^{j+3} + \dots \right) \mathbf{v}_j^{\text{end } j} \cdot \mathbf{v}_j^{\text{end } j} \end{aligned}$$

case $r \neq s, k = \max(r, s)$

$$\begin{aligned} m_{rs} &= m^k l^{j*} \hat{\mathbf{y}}_k^* \cdot \mathbf{v}_r^{\text{start } k} \\ &\quad + \left(m^{k+1} + m^{k+2} + \dots \right) \mathbf{v}_j^{\text{end } k} \cdot \mathbf{v}_j^{\text{end } k} \end{aligned}$$

A similar derivation for the *nonlinear* terms is:

$$(n_{rs})^i = m^i \mathbf{v}_r^{i*} \cdot \dot{\mathbf{v}}_s^{i*} + \omega_r^i \cdot \dot{\mathbf{H}}^i \quad (4.42)$$

where

$$\omega_r^{i*} \cdot \dot{\mathbf{H}}^i = 0 \quad (4.43)$$

In the planar configuration, no torques due to changes in orientation of the angular momentum vectors of bodies occur.

A new term representing outboard mass, $m_{r,s}^*$, is defined as

$$m_{r,s}^* \triangleq m^{k+1} + m^{k+2} + \dots + m^n \quad (4.44)$$

where

$$k = \max(r, s) \quad (4.45)$$

The inertia and momentum scalars for the mass and nonlinear coupling terms matrices are then:

$$m_{r,s} = m_{r,s}^* \mathbf{v}_r^{\text{end } n} \cdot \mathbf{v}_s^{\text{end } n} + m^k (l^* k \hat{\mathbf{y}}_k^*) \cdot \mathbf{v}_r^{\text{end } n} \quad (4.46)$$

and if $j = r = s$

$$+ m^j (l_{j*})^2 + I_{zz}^{j/j*}$$

similarly, the expression for the nonlinear coupling term matrix scalars is:

$$n_{r,s} = m_{r,s}^* \mathbf{v}_r^{\text{end } n} \cdot \dot{\mathbf{v}}_s^{\text{end } n} + m^k (l_k^* \hat{\mathbf{y}}_k^*) \dot{\mathbf{v}}_r^{\text{end } n} \quad (4.47)$$

if $s > r$

$$- m^k (l_k^* \hat{\mathbf{x}}_k^* u_k) \mathbf{v}_r^{\text{end } n}$$

if $r > s$

4.5.3 Modifications for Free-Flying Dynamic Systems

If a system is free-flying then it is necessary to introduce 2 extra linear velocity degrees of freedom and one extra angular velocity degree of freedom. It is typical to introduce these at the body considered as the 'base' of the robot, and let the rotational degree of freedom make the body itself act as a rotational link. It changes the formulations discussed in the sense that body numbers i and generalized speed numbers r are no longer the same: $r = i + 2$.

4.5.4 Modifications for Branching Chain Structures

If kinematic chains have branches, it becomes necessary to take each branch into account when evaluating the kinematic and dynamic terms. The body numbers i and generalized speed numbers r will no longer be the same. The recursion rules must be considered in light of ordering along the chain rather than explicit sequence indices i and $i - 1$.

4.6 Implementation of an Automated Computed-Torque Control System

4.6.1 Motivation: Dynamical System Configurations

In a system consisting of a set of connected bodies, every change in configuration results in different dynamic characteristics. These changes may be due to the addition or removal of bodies, or constraints of motion bodies may place on each other. In mathematical terms, each of these configurations has its own set² of equations of motion that can be used to predict motions of the system. In the problem under study, a free-flying robot with two arms and a payload, the robot-payload system can be in various configurations. The configurations of interest in this research are the following two: (1) both arms are operating independently and (2) both arms are grasping a payload. Many additional variations including the robot gripping onto a rigid object with one arm, and gripping a fixed object with one or more arms, are also possible, as is an object with flexible dynamic properties.

The problem with having multiple configurations is that distinct sets of computed-torque control equations need to be derived and implemented. Manual derivation is not a desirable solution³; it requires significant amounts of the analyst's time and is inherently susceptible to error. The dynamic system under study, a dual-arm satellite manipulator model, is essentially a serial chain of rigid bodies that undergoes only minor changes (in terms of structure) when it grasps an object: chains become longer, or become closed. The equations of motion of a chain system have a certain form as discussed previously in

²Actually, many sets, but they are equivalent.

³The author derived the equations of motion for the case of the 2D free-flying robot with two two-link arms (no payload, no constraints) using Kane's method in 30 pages.

4.6. Implementation of an Automated Computed-Torque Control System 67

this chapter, and the addition of extra links to the system are consistent with this form, although a different set of equations of motion does result. Rederiving the system equations of motion for each new configuration can be avoided by evaluating the terms of equations of motion and control *numerically* at run-time using *algorithms* based on the recursive structure of the kinematics.

4.6.2 The RD Automated Computed-Torque Control Computer Program

The **RD** computer program implements the two dimensional recursive kinematics and dynamics algorithms, and takes advantage of the simplifications available in 2D. A partial listing of the computer code is presented in appendix C. The **RD** computer program constructs and evaluates computed-torque controllers and dynamics simulation equations for rigid-body manipulators with revolute joints from specifications of the masses and inertias of the bodies in the chain as well as their interconnections.

The computer code for the recursive kinematics implements equations 4.1, 4.2, 4.3 to determine positions and velocities on outwards recursions along the kinematic chains. The partial velocities are evaluated according to the rules set forth in equations 4.9 and 4.31. The computer code for the recursive inverse dynamics implements equations 4.15 through 4.24.

The Jacobian matrix and its derivative are formulated using equations 3.6 and 3.8, for ordinary endpoint specifications, equations 3.16, 3.19, for linear momentum specifications, equations 3.17, 3.20 for angular momentum specifications, and equations 3.26 and 3.28 for dynamic constraints.

The simulation code makes use of equation 2.5 or equation 2.6, depending on whether or not dynamic constraints exist in the system. The elements of the mass matrix and the nonlinear terms are evaluated using equations 4.46 and 4.44.

Commented listings for the kinematics and inverse dynamics computer code are presented in appendix C.

4.6.3 Aspects of the Recursive Dynamics Program

This program addresses several issues present in the computed-torque control and simulation of rigid multibody systems. In particular, the dynamic system needs to be defined, the signal inputs and outputs need to be defined, and the control objectives need to be defined. When these items are defined, it is possible both to control and simulate the dynamics of a rigid multibody system. The **RD** program needs to be given a system description computer file prior to use. This system description file consists of five parts: a description of the dynamic system, a description of input and output signals-of-interest (such as positions and velocities and some forces), a description of accelerations of interest, and a description of the control objectives. Comments may be placed in the file, if preceded by a %, in order to increase legibility. Two samples of configuration files are included below: one for a free-flying robot with one two-link arm. Another configures **RD** to control and/or simulate a two-armed fixed-base robot doing cooperative manipulation.

Multibody Dynamics

The first component of the configuration setup file is the description of the multibody system. The syntax uses a start token 'DynamicSystem' and end token 'EndDynamicSystem' to demarcate this section.

A system can be either free-flying or fixed-base, depending on whether the first body is designated as 'FreeFlying' or a 'Body' attached to the 'Inertial' frame at some location. The bodies present in the system are described by their names, where they attach to a previous body, the location of their center-of-mass, their mass and inertia about their center-of-mass. The revolute joints connecting bodies can be named.

Endpoints can be defined and named for future reference in the signals-of-interest, accelerations-of-interest, or control sections. Manipulated bodies can be defined similarly to other bodies, except they have two special properties: they have two connection points to previously defined bodies in the system, and they are eligible to be squeeze-force controlled in the control objectives section. The placement of an object in the system causes **RD** to automatically generate the constraints necessary in the Jacobian, and/or automatically solve the augmented constrained equations of motion.

Signal Inputs

This component of the configuration setup file is optional: the description of input signals-of-interest. The syntax uses a start token 'Inputs' and end token 'EndInputs' to demarcate this section.

Inputs can consist of positions, velocities and forces. Inputs are used by **RD** to substitute its estimate of quantities with measured values, allowing **RD** to accurately infer these quantities further out along the robot manipulator. For example, measured positions can be used to yield a better estimate of grasped object position than kinematically inferred position starting from the base of a robot. **RD** allows the substitution of measured position or velocity values anywhere along a chain. Measured forces can be used to determine the squeeze force on an object from real data.

Signal Outputs

This component of the configuration setup file is optional: the description of output signals-of-interest. The syntax uses a start token 'Outputs' and end token 'EndOutputs' to demarcate this section.

Outputs can consist of positions, velocities and forces. Outputs are computed by **RD** in the course of performing kinematics. The ability to have **RD** compute positions and velocities of arbitrary points in the system (they must have been previously named) allows any kinematically relevant signal to be observed by an external program. It can be used to observe positions or velocities of unmeasured points, or the squeeze force on an object.

Acceleration Outputs

This component of the configuration setup file is optional: the description of dynamic signals-of-interest. The syntax uses a start token 'DynamicOutputs' and end token 'EndDynamicOutputs' to demarcate this section.

The dynamic outputs are evaluated in the process of doing inverse dynamics for computed-torque, or when doing numeric simulations using the equations of motion. Accelerations of points and angular accelerations of bodies can be observed.

Control Objectives

This component of the configuration setup file, the description of the control objectives, is necessary when computed-torque control is to be done. If only simulation is to be done, this component need not be included. The syntax uses a start token 'Control' and end token 'EndControl' to demarcate this section.

The control objectives can include accelerations of (previously named) points, angular acceleration of bodies, system momentum, and object squeeze force. **RD** will check to ensure that sufficient control objectives have been defined for the order of the system – to make sure the resultant Jacobian matrix is square. If this is not true, an error message will be printed.

4.6.4 Sample System Description Files

The following two files describe a free-flying robot with a two-link arm, and a fixed-base robot using cooperating arms to the **RD** program. The sections of this file, covered in the previous paragraphs, serve to describe not only a dynamical system, but the computed-torque control system, measurement update points and points of interest in the system.

This file describes a free-flying robot with one two-link arm.

```
%      Sample Configuration File for RD
%      One-Armed, Free-Flying Robot under Endpoint and Momentum Control
%      Dynamic System Description (all units in SI - kg, m, s)
%
DynamicSystem

FreeFlyingBody "Robot Body"    % Body of Robot
0.0 0.05        % center-of-mass offset in local frame x,y in m
10.0            % mass of base in kg
3.2            % Izz inertia of base in kg-m2

Body           "Upper Arm"      % Upper Arm
AttachedAt     0.1842 -0.1842 % joint attachment point in m
JointType      Revolute         "Shoulder"
0.059 -0.002   % center of mass location of upper arm in m
1.92          % mass of arm in kg
0.02          % Izz inertia of upper arm about center of mass in kg-m2
```

4.6. Implementation of an Automated Computed-Torque Control System 71

```

Body          "Lower Arm"      % Lower Arm
AttachedAt    0.30 0.0          % joint attachment point in m
JointType     Revolute         "Right Elbow"
0.20 0.0      % center of mass location of fore arm
0.34          % mass of fore arm
0.012         % Izz inertia of fore arm about cm

Endpoint      "Endpoint"
AttachedAt    0.30 0.0          % endpoint attachment point in m

EndDynamicSystem

% These quantities are of interest to the error controllers
%
Outputs
    Position   "Endpoint"
    Velocity   "Endpoint"
EndOutputs
%
% These accelerations are of interest to the analyst
%
DynamicOutputs
    Acceleration "Endpoint"
EndOutputs

% These are the set of control objectives
%
Control
    LinearMomentum
    AngularMomentum
    Acceleration "Endpoint"
EndControl

```

This file describes a fixed-base cooperating arm robot. Note that the closed-chain constraint is automatically computed and solved once the points of closure are defined on the object. The tightness of the numerical relaxation of this constraint is set at run time - see the matlab code in appendix D.

```

%      Sample Configuration File for RD
%      Two-Armed, Fixed-Base Robot with Object Control
%      Dynamic System Description (all units in SI - kg, m, s)
%

```

DynamicSystem**Chain**

Body "C" % Upper Right Arm
AttachedAt 0.0 -0.5 % location of arm 1 base (in inertial space)
JointType Revolute "Right Shoulder"
 0.5 0.0 % center of mass location of upper arm
 5.0 % mass of upper arm
 0.1 % Izz inertia of upper arm about cm

Body "D" % Lower Right Arm
AttachedAt 0.6 0.0 % joint attachment point
JointType Revolute "Right Elbow"
 0.20 0.0 % center of mass location of fore arm
 4.0 % mass of fore arm
 0.08 % Izz inertia of fore arm about cm

Endpoint "Right Endpoint"
AttachedAt 0.6 0.0 % endpoint attachment point
EndChain

Chain

Body "E" % Upper Left Arm
AttachedAt 0.0 0.5 % location of arm 2 base
JointType Revolute "Left Shoulder"
 0.5 0.0 % center of mass location of upper arm
 5.0 % mass of upper arm
 0.1 % Izz inertia of upper arm about cm

Body "F" % Lower Left Arm
AttachedAt 0.6 0.0 % joint attachment point
JointType Revolute "Left Elbow"
 0.20 0.0 % center of mass location of fore arm
 4.0 % mass of fore arm
 0.08 % Izz inertia of fore arm about cm

Endpoint "Left Endpoint"
AttachedAt 0.6 0.0 % endpoint attachment point

Object "Object"% Manipulated Object
AttachedAt 0.6 0.0 % gripper port 1
 0.0 0.2 % center of mass location of object
 12.0 % mass of object

4.6. Implementation of an Automated Computed-Torque Control System 73

```

0.2          % Izz inertia of object about cm
Constraint    "Right Endpoint"
0.5 0.0      % constrained endpoint

Endpoint      "Object Point"
AttachedAt    0.0 0.25      % object's center
EndChain

EndDynamicSystem

% These quantities are set to measurements at the appropriate place in the dynamic
%
Inputs
    Position    "Right Endpoint"
    Position    "Left Endpoint"
    Force       "Right Endpoint"
    Force       "Left Endpoint"
EndOutputs

% These quantities are of interest to the error controllers
%
Outputs
    Position    "Right Endpoint"
    Velocity    "Right Endpoint"
    Position    "Left Endpoint"
    Velocity    "Left Endpoint"
    SqueezeForce "Object"
EndOutputs

% These are the set of control objectives
%
Control
    Acceleration    "Object Point"
    AngularAcceleration    "Object"
EndControl

```

4.6.5 Computational Cost

Table 4.1 shows the computational cost in floating-point operations⁴ involved in the **RD** computed-torque controller. The computational cost of solving the system's Jacobian matrix equation

⁴Addition, subtraction, division and multiplication each count as one operation.

is high because it involves solving an equation of the form

$$A_{n \times n} x_{n \times 1} = b_{n \times 1} \quad (4.48)$$

The solution of this matrix equation for the vector x involves $\mathcal{O}(n^3)$ floating-point operations. The solution of this equation is the most computationally intensive component of the computed-torque control technique.

	Kinematics: Operations in Setup
Linear Momentum Setup	$4(n-2)$
Angular Momentum Setup	$13(n-2)$
	Kinematics: Operations per Link
Free-Flying Base	26
Basic Manipulator Link	20
Endpoint	18
Manipulated Object	36
Linear Momentum	28
Angular Momentum	$6+8k$
	System Jacobian Solution
Solution to Jacobian Equation	$(\frac{2}{3}n^3 + 2n^2 - \frac{2}{3}n)$
	Inverse Dynamics: Operations per Link
Free-Flying Base	19
Basic Manipulator Link	21
Endpoint	10
Manipulated Object	29
For each branch in chain	3

Table 4.1: **Number of Floating-Point Operations in RD computed-torque control**

There are n bodies in the system. Note that the computations for angular momentum involve information from the previous k links in the chain.

4.6.6 RD Implementation Limits

The implementation of **RD** used in this research is limited to 10 degrees of freedom in order to keep the memory requirements of the program low. Keeping this limit low also has a slight impact on **RD**'s speed of operation. This limit was not exceeded by the dynamic systems investigated in this thesis.

4.6.7 Verification of Correctness

The correctness of the algorithm for dynamics modeling was confirmed in two ways: by checking the conservation of the system's Hamiltonian and by comparison to results of SDEXACT [26] simulations.

A self-check can be made by observing the energy of the system over a simulation run. Also, as time step size is decreased, the variation of energy over time decreases in greater proportion.

4.7 Summary

The special properties of kinematic chains have been used to formulate recursive equations useful for performing computed-torque control and also for performing dynamic simulation. These recursive equations have been implemented in a computer program, **RD**, that can be used to simulate the motions of a system of rigid bodies, and it may also be used to do computed-torque control. A simple system description file is all that is needed to start investigating the dynamics of systems of multiple rigid bodies.

2

Chapter 5

Experimental Hardware

This chapter discusses the robot used to perform the experimental work described in chapter 6 and 7. The requirements that the experimental free-flying robot model had to meet in order to be a viable experimental platform are discussed. An overview of the robot's mechanical, electrical and computer system components is presented.

5.1 System Requirements

The central control problems being studied in this research are (1) independent manipulator endpoint control, and (2) cooperative manipulation of an object from a free-flying robot. In order to model a real space robot accurately, this experimental space robot model must have various characteristics in common with it. The experimental robot should be free-flying, with negligible external forces and torques, it should have multiple manipulator arms, and it should be able to use its arms cooperatively to manipulate a free-flying payload. In order that the model developed for control and simulation is applicable, it is also necessary that the robot not have flexibility in the links or drive train anywhere near that of the bandwidth of the control system. Friction effects in the drive-train and base motion have been made as small as possible.

This experimental apparatus was designed¹ so as to model realistically a free-flying robot, such as NASA's proposed Orbital Maneuvering Vehicle. This free-flying robot model

¹The design was undertaken by Marc Ullman and the author, with help from Gad Shelef in the design and fabrication of the manipulator arms.

is used to test control algorithms developed in this research project, and is expected to be used in future research.

5.2 Design Approach

A modular design approach, allowing rapid disassembly, repair, modification, and reassembly, was taken. Modularity offers not only ease of use for our generation of experiments, but allows simpler upgrade paths to be taken for the needs of future researchers. New subsystems can be added or older technology items (particularly computers) can be replaced as the state-of-the-art improves. The space robot model was designed to be autonomous in order to model faithfully the dynamics of a space-borne free-flying robot. Self-contained systems for power, sensing and computing allow the model robot to function without cumbersome tethers to the outside world.

5.3 Robot System Overview

The robot model used in the experiments was designed to be able to function autonomously, such that it may perform manipulation and navigation tasks with the same kind of freedoms enjoyed by a true space-based robot. The robot is fully self-contained, carrying its own pressurized air for flotation and thrust (although no thrust is used in these experiments, it is intended for use in others), batteries for electrical power, two manipulator arms for catching and positioning payloads, and analog electronics for sensing and actuation, as well as an on-board computer for the control software. All these subsystems fit into the robot, which measures 500 mm in diameter and 800 mm in height. The free-flying satellite robot model with two manipulator arms is shown in figure 5.1. Various components making up this robot model will be discussed in the following sections.

It is possible to see, from the top down, the computer layer, the analog electronics layer, the pressurized gas layer for flotation and thrusters, and the manipulator arms with endpoint grippers. The base of the robot masses 50 kg, and has an inertia about its center of mass of 3.2 kg-m^2 . Each of the arms mass 2.2 kg. The mass distribution in the robot system is presented in detail in table 5.1.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

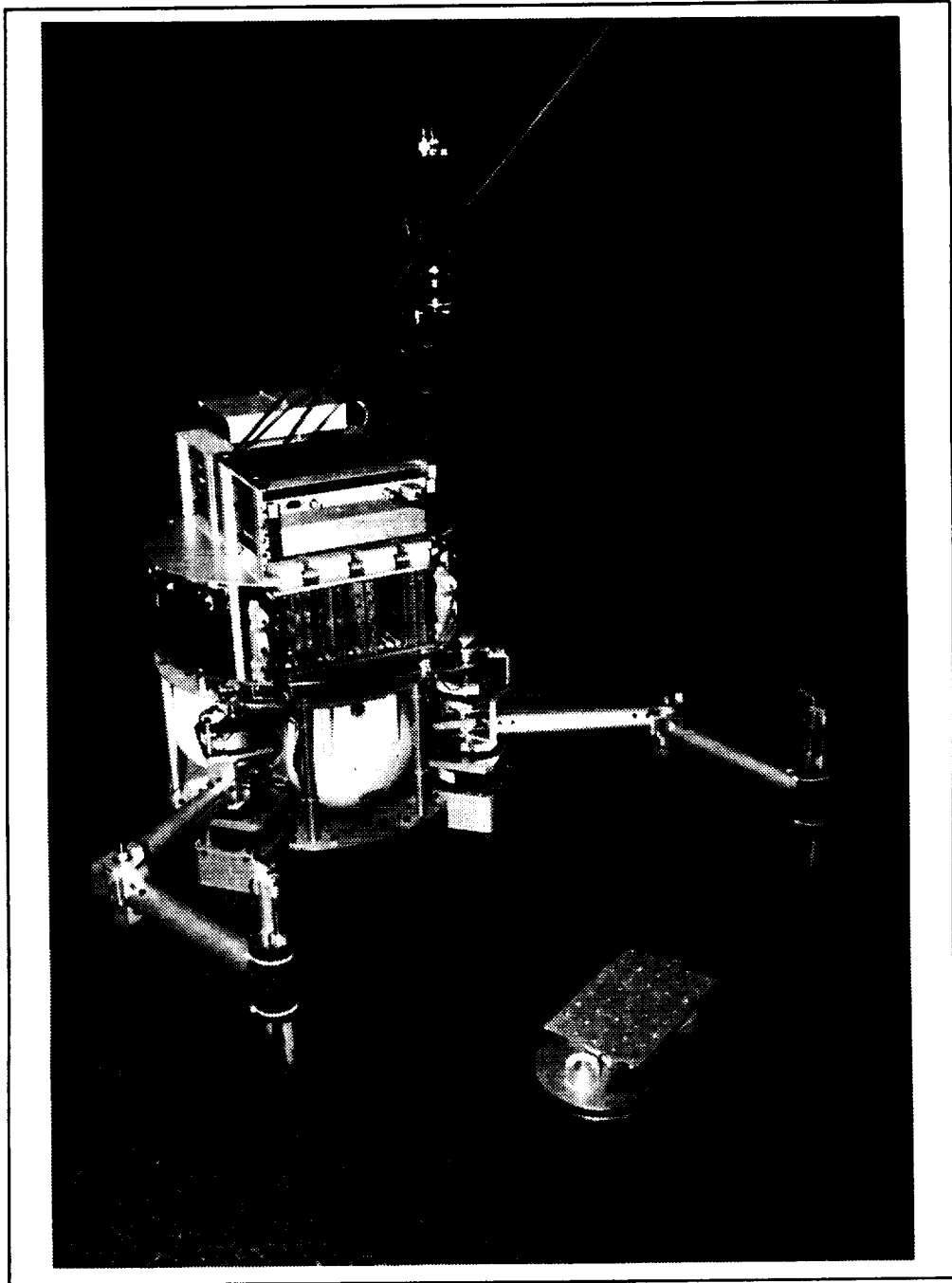


Figure 5.1: **Satellite Robot Model**

The Satellite robot model is a fully autonomous system, with on-board compressed air for propulsion and flotation, on-board batteries for power, and an on-board computer for controllers.

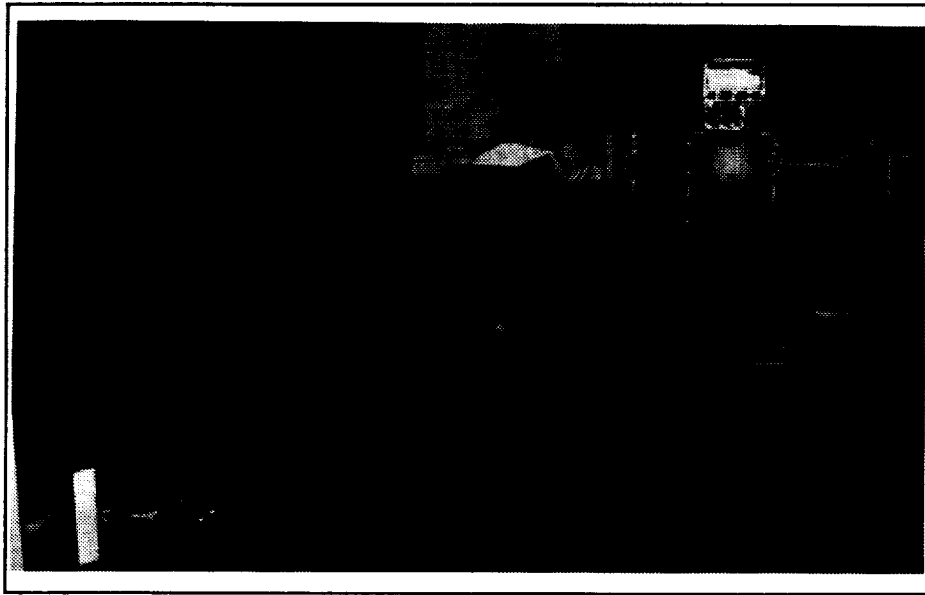


Figure 5.2: Granite Surface Plate

This 9 foot x 12 foot granite surface plate affords a large area workspace on which to perform experiments.

5.4 Air-Bearing

The experimental robot requires the existence of a very-low-friction bearing for faithful simulation of the drag-free space environment. Previous research by Alexander [1] has found air bearings to be ideally suited to these needs. The principals of operation of this type of bearing are discussed by Alexander [1] and Rehsteiner [24]. The air bearing is formed by a laminar flow of air forced outward between two smooth, flat surfaces. The lower surface of the bearing is a large granite surface plate. The plate is a rigid, dimensionally controlled surface that can tolerate a heavy robot and not bend. Figure 5.2 shows a picture of the granite surface plate used in this research project.

The upper surface of the bearing is a 20 inch diameter, 1/16 inch aluminum plate bonded to a piece of 1 inch Hexcel in order to give it stiffness, and then lapped in order to make it smooth and flat. The operation of this air bearing - showing air flow, is shown in figure 5.4. The thickness of the air bearing measured was approximately 2 thousandths of inch at a feed pressure of approximately 1 psi. This is not a critical figure: higher pressures

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH

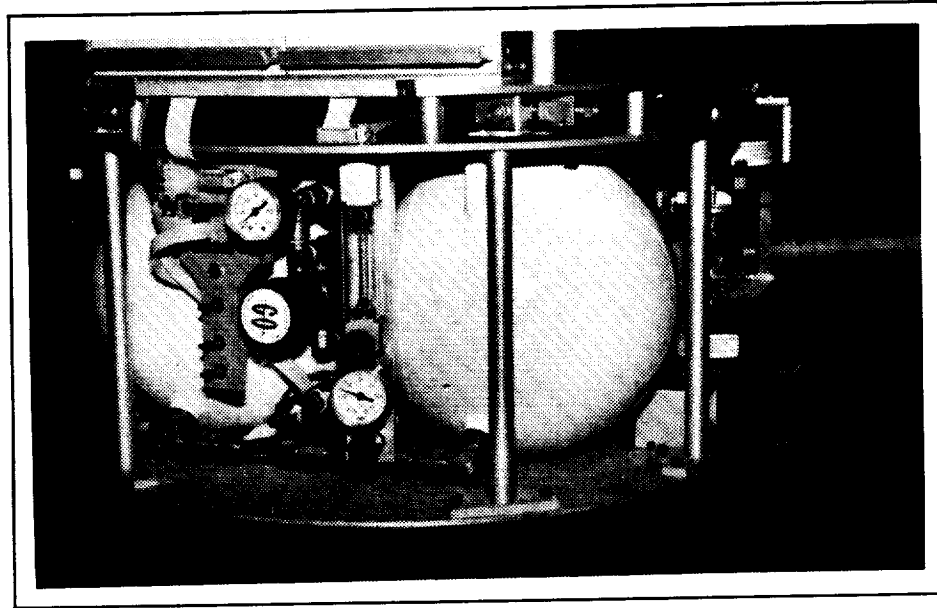


Figure 5.3:

Pressurized Gas Layer

Contains three gas pressure vessels, high and low pressure regulators, air-bearing flow control and gas pressure sensors.

and flow rates assure operation when greater quantities of dust exist on the surface plate. The supply of pressurized gas to this bearing is controlled by a valve, and comes from a low-pressure regulator. Pressurized gas is stored in spherical tanks located in the lower layer of the robot. A photograph of this pressurized gas subsystem is presented in figure 5.3. The design and manufacture of this subsystem is discussed in the thesis of Ullman [35].

5.5 Manipulator Subsystem

The manipulator arms used in this robot² are similar to the ones used in the research of Schneider [29]; however, in these experiments, the arm links are 100 mm shorter (250 mm in length), so that the manipulators can exert greater endpoint forces in a decreased workspace. The manipulator arm was designed as a SCARA manipulator and can position the endpoint in 2D. The link joints use high-quality bearings to achieve low friction

²The arms were conceptually designed by the author and Stan Schneider, and mechanically designed and manufactured by Gad Shelef.

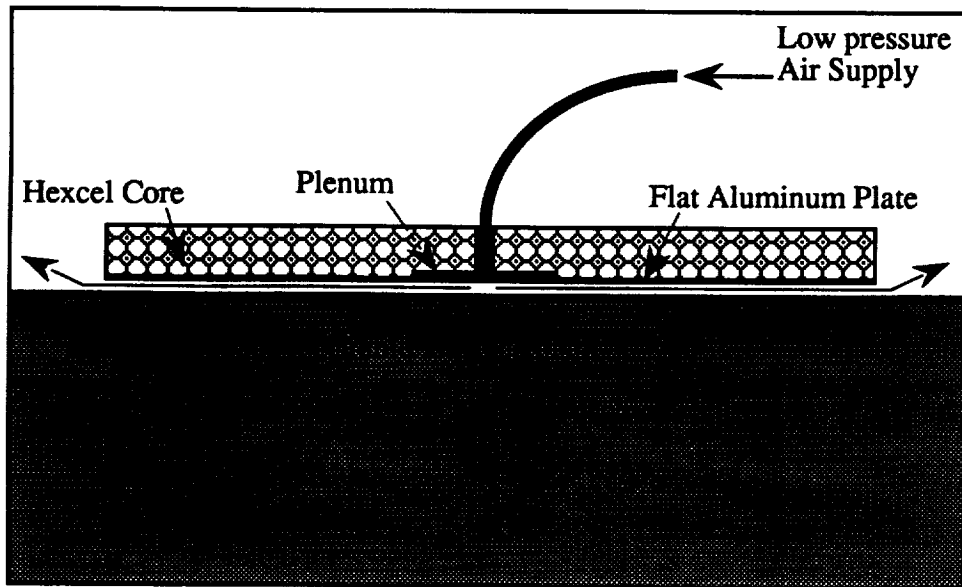


Figure 5.4:

Air Bearing

The air bearing operates via a very thin layer of laminar flow low-pressure air.

ORIGINAL PAGE

BLACK AND WHITE PHOTOGRAPH

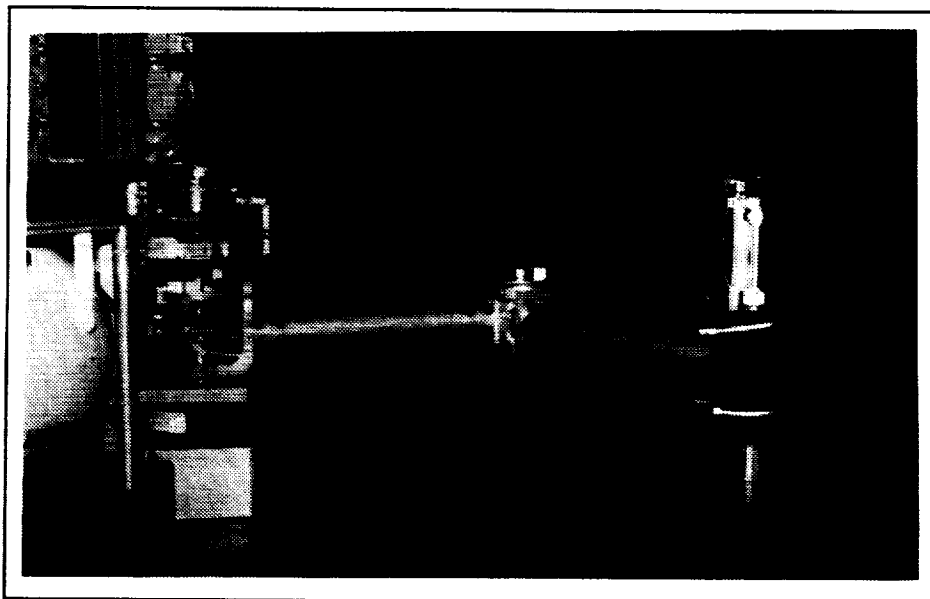


Figure 5.5:

Manipulator Arm

Two of these manipulator arms are mounted on the robot. High-quality bearings and brushless DC torque motors are used to achieve low friction to motion.

to motion, while the motors used are Aeroflex brushless DC torquers (V40Y-6H at the elbows, V40Y-5H at the shoulders) capable of delivering torque smoothly, with low friction and extremely low ripple torque. The torque motors are located at the base of the arm. This location offers two major advantages: the arm links have low inertia for fast response, and the center of mass of the robot vehicle does not shift far from the center of the air bearing with arm displacements - so there is little side thrust due to imbalances in loading the air bearing. One of the manipulators is pictured in figure 5.5.

At the arm endpoints are force-sensing grippers. The manipulator arm endpoint has a gripper assembly that can be pneumatically lowered and raised to capture and release payloads. Applied forces are sensed using strain gauges mounted at the bottom of the plunger mechanism. A strain-gauge-signal conditioning board³ provides conditioned signals indicating the forces at the endpoint. The schematic diagram of this signal conditioning board is presented in appendix B. An infrared LED located at the top of the gripper can be seen by the vision system and used to track the endpoint position. The plunger terminates in a bearing surrounded by an O-ring. The bearing ensures that no vertical-axis torque is exerted at the endpoint, and the O-ring provides a snug fit into gripper ports located on objects to be grasped.

5.6 Sensor Subsystems

5.6.1 Vision System

An overhead vision system is used to determine position and orientation of the robot vehicle and the payload, and also allows us to measure the robot manipulator's endpoint position. It uses a camera mounted over the operating surface, a custom VME board that processes the video signal and provides data to a computer, and vision software that provides estimates of point locations and body orientations for control programs. The camera, a Pulnix model TM440S, is shown in figure 5.6. This VME-bus vision board's⁴ principle of operation is as follows: endpoints are each marked with an infrared LED, and

³The board was designed by Yosi Druker and built by Godwin Zhang

⁴The vision board was designed, constructed and documented by Vincent Chen [5] and is based on earlier, simpler systems developed by Alexander [1] and the author.

ORIGINAL PAGE
BLACK AND WHITE PHOTOGRAPH



Figure 5.6: **Global Vision Camera**

This camera is mounted 2 meters above the granite table and provides a scan image of the workspace 60 times per second.

bodies are marked with three infrared LEDs. A signal coming out of a video camera is used by the vision board to determine where bright areas are in the camera's view. These bright areas are registered and tracked in vision server software⁵ which tracks named points and bodies, and provides estimates of position and velocity (and angular orientation and rate for bodies) to the control systems.

5.6.2 Joint Angle Sensors

Manipulator joint angles are measured at the motors by analog RVDT's (rotational variable differential transformers). They are Pickering model 23501-0, and have a linearity of within 1% over the range sensed (150 deg). An RVDT signal conditioning board that converts the RVDT sensor outputs to angle and band-limited angular rate for the control system was designed by Tzoor Friedman. Its circuit diagram is included in appendix B. The RVDT circuit board provides an excitation signal for the RVDTs and decodes their output.

⁵The vision server software was developed and documented by Stan Schneider [29].

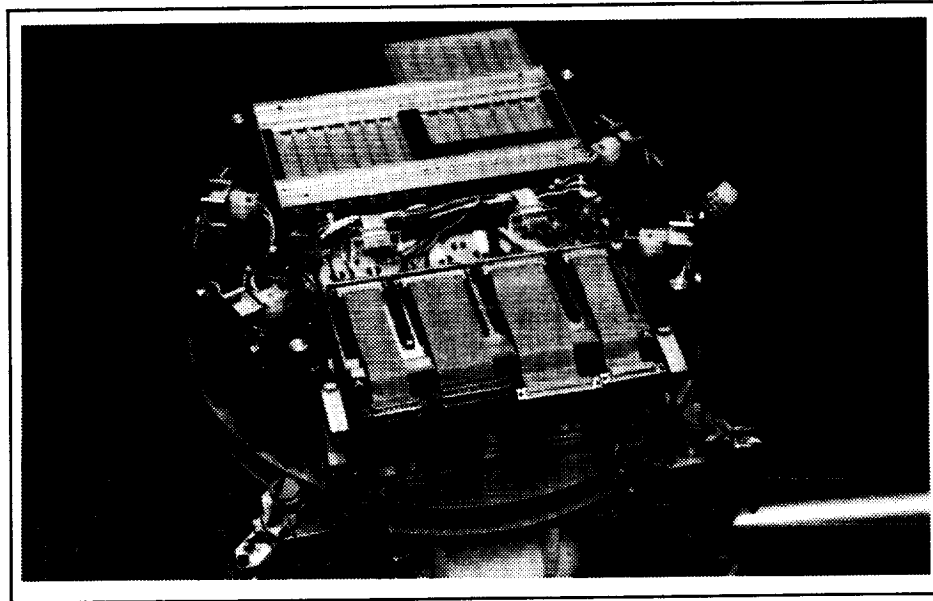


Figure 5.7:

Analog Electronics Layer

The analog electronics layer comprises the two battery packs, power converters and protection circuitry, the circuit boards for power control, battery charging, RVDT conditioning, force sensing and safety disconnection.

5.6.3 Body Angular Rate Sensor

An angular rate sensor is mounted on the body of the robot, and is used to provide an estimate of its angular rate about the z (vertical) axis. The angular rate sensor is a Watson Industries C131 1AV with a sensitivity of 10deg/sec-V. Its output is run through a first order 15 Hz RC filter in order to reduce high-frequency noise prior to sampling. Due to drift and bias, it is not suitable for long-term estimates of the robot's orientation: this is provided by the vision system. The sensor is documented in the thesis of Jasper [11]. The circuit diagram depicting this hookup is shown in appendix B.

5.7 Analog Electronics Layer

5.7.1 Power Subsystem

The requirement for minimal external interference with free-flying robot motion makes it difficult to feed power to the robot from an external source. Some free-flying vehicles use power feed umbilicals that hang from a tracking platform. Such a rig is complex to design and build, and in the long run is not a viable option because of the problems caused if one were to operate two robot vehicles in the same vicinity. Hence, the experimental space robot model has an autonomous power system that allows the experiment to function for extended periods of free flight – 45 minutes to 3 hours depending on whether extensive (power consuming) manipulation tasks are performed.

The power system mimics the functionality of spacecraft power systems. It includes rechargeable NiCad batteries, battery switching control, a raw power bus, a source of external power for recharging batteries (solar cells or a nuclear source could be used in space), power converters and protection circuitry.

When tethered, the vehicle is powered from the external power source to minimize drain on the batteries, and when free-flying, it is powered by the batteries. The batteries are 12 Volt VARTA RSH-7 fast-charge, high discharge-rate NiCads. Due to finite charge in the batteries, they will need to be recharged or replaced often. One luxury of an earth-bound experiment is that battery packs can be replaced during extended untethered operation. To ensure that vital on-board systems, such as the computer, remain functional, the power system has been designed so that power delivery to on-board systems is not interrupted when on-board energy sources are replaced, or external power sources engaged or disengaged.

5.7.2 Power Management and Distribution

A schematic overview of the power distribution system of the experimental robot is shown in figure 5.8. Circuit diagrams to the various components are included in appendix B.

Power is controlled via the power control unit (PCU) by a master power switch, and a battery switch for each of the two batteries. Power can be switched onto the system's raw power bus from three power sources: the two batteries and external power. Multiple sources can be switched on at the same time thereby ensuring continuous power supply in

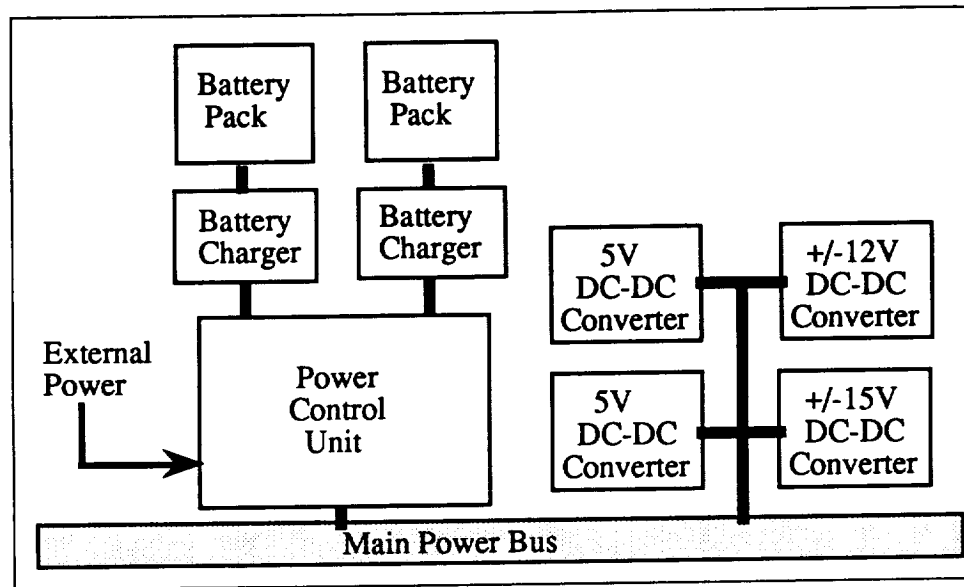


Figure 5.8: **Power System Overview**

The power system accepts external power when docked, and can use it to charge the on-board batteries. During free-flying operation, the two batteries power the robot. Regulated power for the analog and digital circuitry is provided by DC-DC power converters.

the event of the removal of a source, such as unplugging external power or a battery pack. The PCU has sensors that allow the computer to read the status of the power system. The PCU also has override functions, which allow the computer to automatically switch fresh batteries on and old batteries off the power bus. Batteries are protected with a 15 Amp fuse to limit discharge, and diodes to prevent back-charging off the bus, and with a 2 Amp fuse to protect the battery from excessive charge currents. The raw power bus is unregulated $\pm 12\text{VDC}$ at up to 15 Amp, which is used for the motor drivers and two sets of power converters/conditioners.

One set of power converters (two Computer Products 24S05/50K3 DC-DC converters) is used to provide regulated 5 V power to the computer and digital electronics. Another pair (Computer Products 24D15/60K3 and 24D12/60K3 DC-DC converters) provide regulated $\pm 12\text{ V}$ and $\pm 15\text{ V}$ power to the analog electronics section (e.g. sensor electronics, etc.). Voltage fluctuations due to changing loads and power spikes on the raw power bus will not be seen on either the computer or analog electronics regulated power busses.

5.8 Computer and Communications Architecture

During experimental runs, three computers are used: An on-board computer does the control functions, an off-board real-time system does vision system processing, and a SUN workstation does data logging.

Figure 5.9 shows a schematic overview of the organization of the computer networks used to interconnect these computers. This computer system architecture is an outgrowth of the work of many students in the Stanford Aerospace Robotics Laboratory, and is documented in the thesis of Ullman [35].

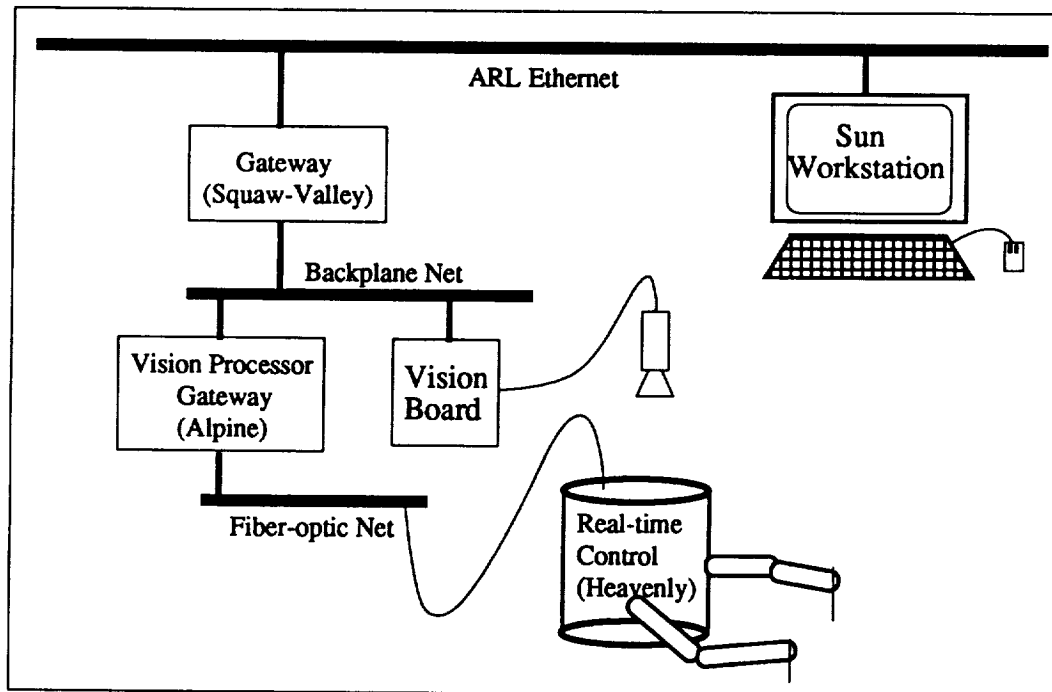


Figure 5.9: **Software Development Computer Network**

The Stanford Aerospace Robotics Laboratory computer network allows a SUN software development system to communicate with real-time computer systems that perform vision processing and robot control. The gateway processors isolate the real-time systems from traffic on the main computer network.

The real-time computer is a Motorola VME 127-1 25 MHz 68030 single board computer, running the VxWorks real-time operating system. The analog-to-digital converter used was the 16 differential channel Xycom XVME-590 with 12 bits of resolution and a valid input

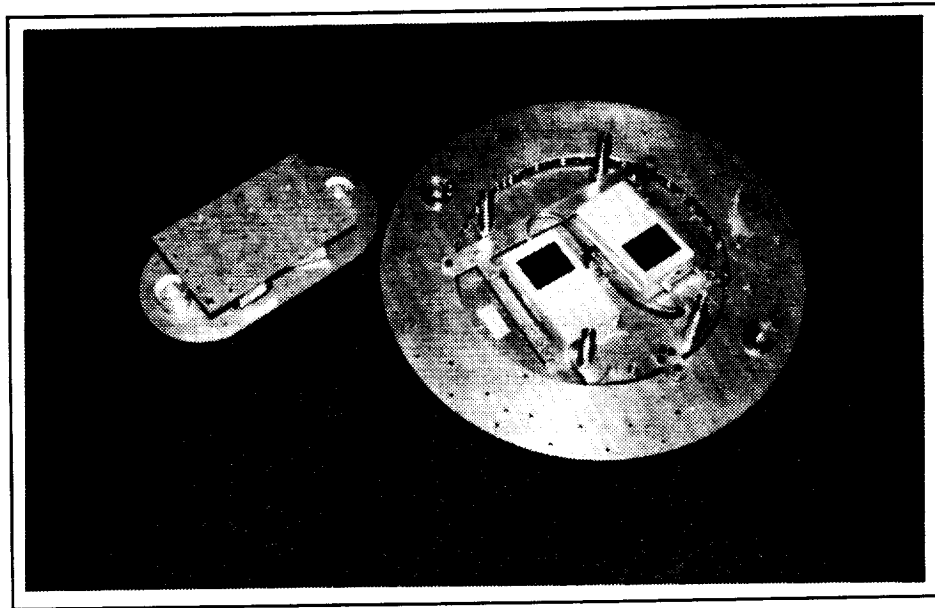


Figure 5.10: **Free-Flying Objects used in Manipulation Experiments**

These two-free flying objects use aquarium pumps to supply their air bearings. Gripper ports allow them to be grasped by the robotic arms.

voltage range of ± 10 V. Differential input sensing and scaling of signals to the order of 5-10 V was used to keep noise and floating-ground effects on measured signals to a minimum. The digital-to-analog converter used was the 4 channel Xycom XVME-595 with 12 bits of resolution and a valid output voltage range of ± 10 V. Differential sensing in the output servo amplifiers was used to minimize the effects of differential ground-plane voltages on inputs to the torque motor amplifiers. Particular care was taken to reduce the effects of induced currents and floating ground plane voltages in this experiment because it has no electrical ground when free-flying, and the power and motor electronics use high currents in a tightly packaged environment.

5.9 Target Vehicles

This experimental work involves the manipulation and control of free-flying objects. Two objects were constructed for this experiment: a light, small object and a larger, heavier

object. Figure 5.10 shows the two free-flying objects that were used in the course of the experiments. The gripper ports – the two plastic cups at either ends of the objects – serve as the connection points for the arm grippers.

5.10 Summary

The experimental apparatus, a two-armed free-flying robot, contains compressed air for thrusters and flotation, batteries for on-board power, and computer and analog electronics for sensing and control. A global vision system allows direct endpoint sensing in an inertial reference frame.

The robot truly does model in 2D the motions one would expect of a free-flying robot. The application of a slight external force impulse can cause it to translate and rotate for a substantial length of time – on the order of a minute if the air-bearing table is clean.

Base Mass	50.0 kg
Base Inertia	3.2 kg-m ²
Left Arm Attachment Point on Base	(148, 148) mm
Left Upper Arm Center of Mass	(59, 2) mm
Left Upper Arm Mass	1.92 kg
Left Upper Arm Inertia	0.02 kg-m ²
Left Lower Arm Attachment on Upper Arm	(305, 0) mm
Left Lower Arm Center of Mass	(200, 0) mm
Left Lower Arm Mass	0.34 kg
Left Lower Arm Inertia	0.012 kg-m ²
Left Endpoint Location on Lower Arm	(295, 0) mm
Right Arm Attachment Point on Base	(148, -148) mm
Right Upper Arm Center of Mass	(59, -2) mm
Right Upper Arm Mass	1.92 kg
Right Upper Arm Inertia	0.02 kg-m ²
Right Lower Arm Attachment on Upper Arm	(305, 0) mm
Right Lower Arm Center of Mass	(200, 0) mm
Right Lower Arm Mass	0.34 kg
Right Lower Arm Inertia	0.012 kg-m ²
Right Endpoint Location on Lower Arm	(295, 0) mm

Table 5.1: Free-Flying Robot Mass and Geometry Parameters

The base's mass and inertia are large compared to the manipulator arms. Coordinates are given in robot segment local coordinates relative to a connection point (0,0): x is along the body toward the next connection point, and y is perpendicular in the right-handed sense. Arm connection points are given relative to the center of the base.

2

Chapter 6

Endpoint and Object Control Experiments

In this chapter, the automated computed-torque control computer program developed in chapter 4 (**RD**) will be used for dynamic simulation of, and experimental computed-torque control of the two-armed, free-flying robot model shown in chapter 5. The simulation runs will verify that the control laws work, generating expectations for experimental results, and thereby show that a simulation model can predict basic behavior of the system. The **RD** computer program is used to model the full dynamics of the free-flying robot, which consists of multiple rigid bodies.

Two experimental demonstrations of control capability are performed using the free-flying robot model: The first is simultaneous manipulator CT endpoint position control of the robot's two arms, even though the robot body can translate and rotate and react to manipulator motion. The second demonstration is cooperative-arm manipulation of a free-flying object. The object is both position controlled and orientation controlled. The squeeze force applied by the manipulators on the object is also controlled. In both cases, no use of external forces (thrusters) or torques (reaction wheels) was made. The robot tended to drift or rotate away from its preferred workspace quite rapidly because of this.

The components of the CT control system: the commanded trajectories, the dynamic system, the computed-torque controller and the error control laws will be discussed first. This is followed by a simulation run and then the experimental results.

6.1.1 Endpoint Trajectories

As listed in table 6.1, the right manipulator endpoint was commanded to hold a fixed position in inertial space, and the left endpoint was commanded to follow a circular trajectory in inertial space of radius 50 mm at 50 rpm. The circular trajectory generator yielded desired position, desired velocity and estimated acceleration feedforward for the endpoints.

	x (mm)	y (mm)
Right Endpoint	0	-100
Left Endpoint	$50 \sin 2\pi \frac{50}{60}t$	$100 + 50 \cos 2\pi \frac{50}{60}t$

Table 6.1: **Trajectory Specification for the Two Endpoints**

The left arm endpoint is to be held fixed in inertial space, the right arm is to follow a circular trajectory in inertial space.

$$\mathbf{p}_{traj}^p = r \begin{bmatrix} \sin 2\pi\omega t \\ \cos 2\pi\omega t \end{bmatrix} + \mathbf{r}^{center} \quad (6.1)$$

$$\mathbf{v}_{traj}^p = 2\pi\omega r \begin{bmatrix} \cos 2\pi\omega t \\ -\sin 2\pi\omega t \end{bmatrix} \quad (6.2)$$

$$\mathbf{a}_{traj}^p = -(2\pi\omega)^2 r \begin{bmatrix} \sin 2\pi\omega t \\ \cos 2\pi\omega t \end{bmatrix} \quad (6.3)$$

6.1.2 Computed-Torque Controller

The automated computed-torque controller program (**RD**), described in chapter 4, will be used both to simulate the dynamics of, and do CT control of the two-armed free-flying robot described in the previous chapter. The program accepts a description of the dynamic system at startup. This description file, presented below for the system under

study, specifies the organization of the dynamic system, the quantities of interest (endpoint information), and the control objectives for use in the CT controller. The organization of this file (including format, commands, and units) was discussed in section 4.6.

```
%      Sample Configuration File for RD
%      Two-Armed, Free-Flying Robot under Endpoint and Momentum Control
%      Dynamic System Description (all units in SI - kg, m, s)
DynamicSystem

FreeFlyingBody "B"      % Body of Robot
0.0 0.0          % center-of-mass offset in local frame
50.0             % mass of base
3.2             % Izz inertia of base

Chain

Body            "C"      % Upper Right Arm
AttachedAt      0.1842 -0.1842 % joint attachment point
JointType       Revolute   "Right Shoulder"
0.059 -0.002     % center of mass location of upper arm
1.92            % mass of upper arm
0.02            % Izz inertia of upper arm about cm

Body            "D"      % Lower Right Arm
AttachedAt      0.3048 0.0    % joint attachment point
JointType       Revolute   "Right Elbow"
0.20 0.0        % center of mass location of fore arm
0.34            % mass of fore arm
0.012           % Izz inertia of fore arm about cm

Endpoint        "Right Endpoint"
AttachedAt      0.2953 0.0    % endpoint attachment point
EndChain

Chain

Body            "E"      % Upper Left Arm
AttachedAt      0.1842 0.1842 % joint attachment point
JointType       Revolute   "Left Shoulder"
0.059 0.002     % center of mass location of upper arm
1.92            % mass of upper arm
0.02            % Izz inertia of upper arm about cm

Body            "F"      % Lower Left Arm
AttachedAt      0.3048 0.0    % joint attachment point
```

```

JointType      Revolute      "Left Elbow"
0.20 0.0                % center of mass location of fore arm
0.34                % mass of fore arm
0.012                % Izz inertia of fore arm about cm

Endpoint      "Left Endpoint"
AttachedAt    0.2953 0.0      % endpoint attachment point
EndChain
EndDynamicSystem

% These quantities are of interest to the error controllers
%
Outputs
    Position      "Right Endpoint"
    Velocity      "Right Endpoint"
    Position      "Left Endpoint"
    Velocity      "Left Endpoint"
EndOutputs

% These are the set of control objectives
%
Control
    LinearMomentum
    AngularMomentum
    Acceleration  "Right Endpoint"
    Acceleration  "Left Endpoint"
EndControl

```

This configuration file describes a free-flying robot consisting of bodies B, C, D, E and F, with two manipulator arm endpoints designated "Right Endpoint" and "Left Endpoint". The endpoint position kinematically determined by **RD** is used by the vision system to locate the endpoints in its field of view. Vision system position information is used by the endpoint controllers. The endpoint velocity determined by **RD** is used in the error controller, rather than using vision data. The control objectives are the accelerations of these two endpoints and control of the system's linear and angular momentum rates. **RD** will automatically evaluate the Jacobian equation and inverse dynamics equations to solve for manipulator torques given these control objectives. It can also, at program command, evaluate and solve the system dynamical equations of motion for simulation.

The complete set of control objectives for this system are as follows: system linear momentum, system angular momentum, and the two endpoint accelerations. The augmented Jacobian technique discussed in chapter 3 is used. The control objectives are explicitly stated here:

$$\mathbf{a}^S \stackrel{\triangle}{=} \underset{3.29}{\begin{bmatrix} \frac{d}{dt}\mathbf{L}^S \\ \frac{d}{dt}\mathbf{H}^{S/S^*} \\ \mathbf{a}^{P1} \\ \mathbf{a}^{P2} \end{bmatrix}} \quad (6.4)$$

Note that the desired momentum rates are specified as zero in this investigation to reflect the conservation of momentum when no thrusters are to be used.

6.1.3 Error Controllers

The error controllers ensure that the endpoints track commanded trajectories. Error controllers (part of the feedback control system) are only necessary if the behavior of the system is not exactly as expected, which of course is always true: to some degree there will exist modeling errors, measurement errors in the state, or unmodeled disturbances.

Figure 6.2 shows that aspects of this particular dynamic system that have not been modeled, the spring and friction forces of wiring in the joints, do contribute to error in the ability to control endpoint position: an error feedback controller is necessary. These forces, although small, are very difficult to model: the wiring and pneumatic tubes in the arms grab and release as the joints are moved, so the forces (moments) they exert are nearly impossible to predict accurately. It is the role of the feedback controller to drive the errors resulting from such unmodeled dynamics to zero.

The control objectives in this experiment are the accelerations of the two endpoints and the system momentum rates. The desired momentum rates were always zero: effectively telling the control system that momentum was to be conserved.

The error controllers for endpoint positions determine a desired endpoint acceleration to cause the error to behave in a known manner. The desired acceleration to respond to endpoint position and velocity errors was computed using the following control law:

$$\mathbf{a}_{desired}^P = K_p(\mathbf{p}_{des}^P - \mathbf{p}^P) + K_v(\mathbf{v}_{des}^P - \mathbf{v}^P) \quad (6.5)$$

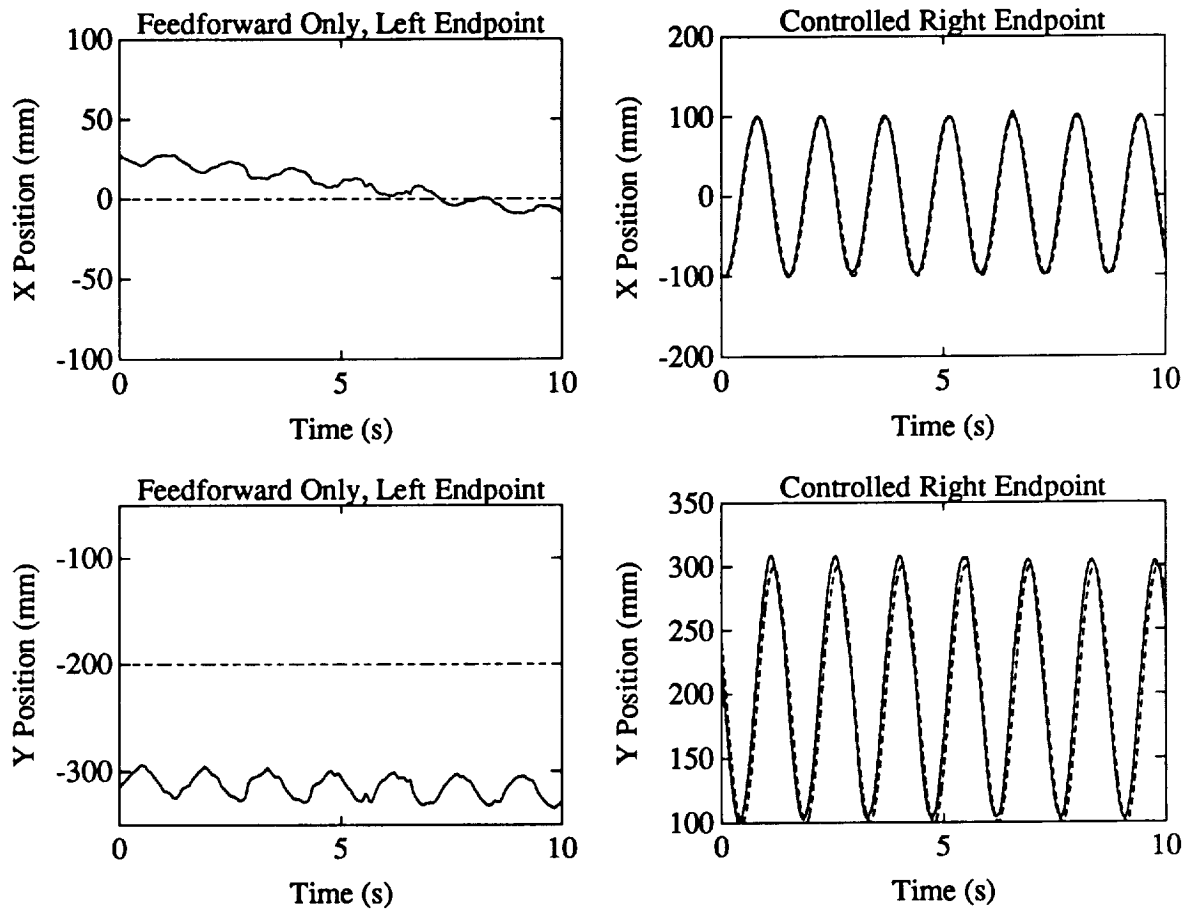


Figure 6.2: **Endpoint Position Control from a Free-Flying Robot: Feedforward versus Feedback**

Experimental data illustrating simultaneous arm endpoint control from a free-flying robot. Solid lines are experimental data, dashed lines are the desired values. The left endpoint is feedforward controlled to a fixed location in inertial space, the right endpoint is feedback controlled to follow a circular trajectory. The sinusoidal errors in the left endpoint position are primarily due to the unmodeled dynamics of arm joint spring forces (due to tubing and wiring), and joint friction forces.

The values for the position gains K_p and velocity gains K_v are listed in table 6.2. The desired response of these second-error position controllers was a damped sinusoid of natural frequency 1.5 Hz with a high damping coefficient of 1.1.

	$K_p (m/s^2)/(m)$	$K_v (m/s^2)/(m/s)$
Right Endpoint	100	22
Left Endpoint	100	22

Table 6.2: **Second-Order Error-Controller Gains**

These gains were used in both arm-endpoint position error controllers.

The desired acceleration specified to the computed-torque controller for the endpoints was computed using the trajectory feedforward acceleration and the error controller:

$$\mathbf{a}_{commanded_{6,3,6,5}}^p = K_p(\mathbf{p}_{des}^p - \mathbf{p}^p) + K_v(\mathbf{v}_{des}^p - \mathbf{v}^p) + \mathbf{a}_{traj}^p \quad (6.6)$$

6.1.4 Simulation Run

An animated simulation of this system with its controller in operation is presented in figure 6.3¹, and shows the two endpoints behaving correctly: one is stationary at (0, -200) mm, and the other is following a circular trajectory about (0, 200) mm. The robot body, although initially at rest, moves in response to manipulator activity. The majority of the base motion in response to manipulator activity is angular: there is little translational motion. In this simulation, the controller and simulation dynamic models were perfectly matched.

6.1.5 Experimental Results

The results of two experimental runs are presented: one that shows the robot's endpoints following their commanded trajectories, and another that shows disturbance rejection. In all cases, the controller's sample rate was 60 Hz². Endpoint positions were measured with the vision system. It was not practical to pseudo-differentiate this position information

¹The matlab script file for this simulation is presented in appendix D.

²Due to the vision system frame rate.

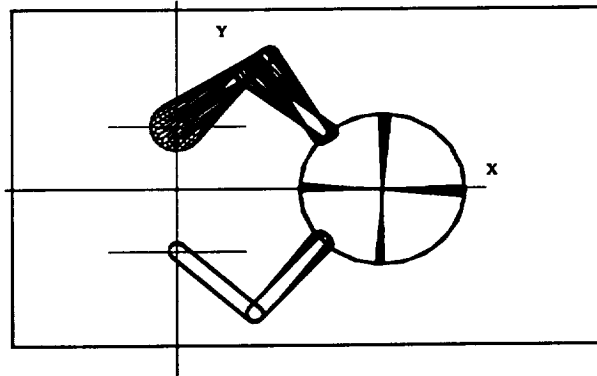


Figure 6.3: **Animated Matlab Simulation Run of Endpoint Control from a Free-Flying Robot**

This is a dynamic simulation of the experimental robot system with an endpoint controller. Notice that robot base motion is primarily angular in nature: arm motion does not result in significant robot base translation.

(by first-differencing) due to quantization and noise. Therefore, endpoint velocities were estimated given the base velocity (from the vision-system base-position measurements), base angular-speed from the angular speed sensor, and manipulator-arm joint rates. This controller consumed 45% of the CPU time on the 25 MHz 68030 real-time microcomputer.

In figure 6.4 the performance of the two endpoint controllers are shown. Errors can be attributed to unmodeled dynamics such as friction in manipulator bearings and spring forces introduced by wiring and pneumatic tubes in the arms, and mismodeling of the sensors and actuators³. The magnitude of these errors depended on the configuration of the robot: for example, as the robot arms moved, spring forces would change and sliding of wiring could cause sudden changes in the spring forces. Unlike with fixed-base robots, these effects are not easily reduced via position integral control. Free-flying robots, by their very nature, do not maintain constant manipulator angles over time when holding a payload stationary. Integral control excels at maintaining accurate stationary positioning of joints in the face of poorly modeled effects of gravity, mismodeled kinematics, and errors due to motor forces and friction forces.

³Gain errors, offsets, nonlinearities and friction effects exist.

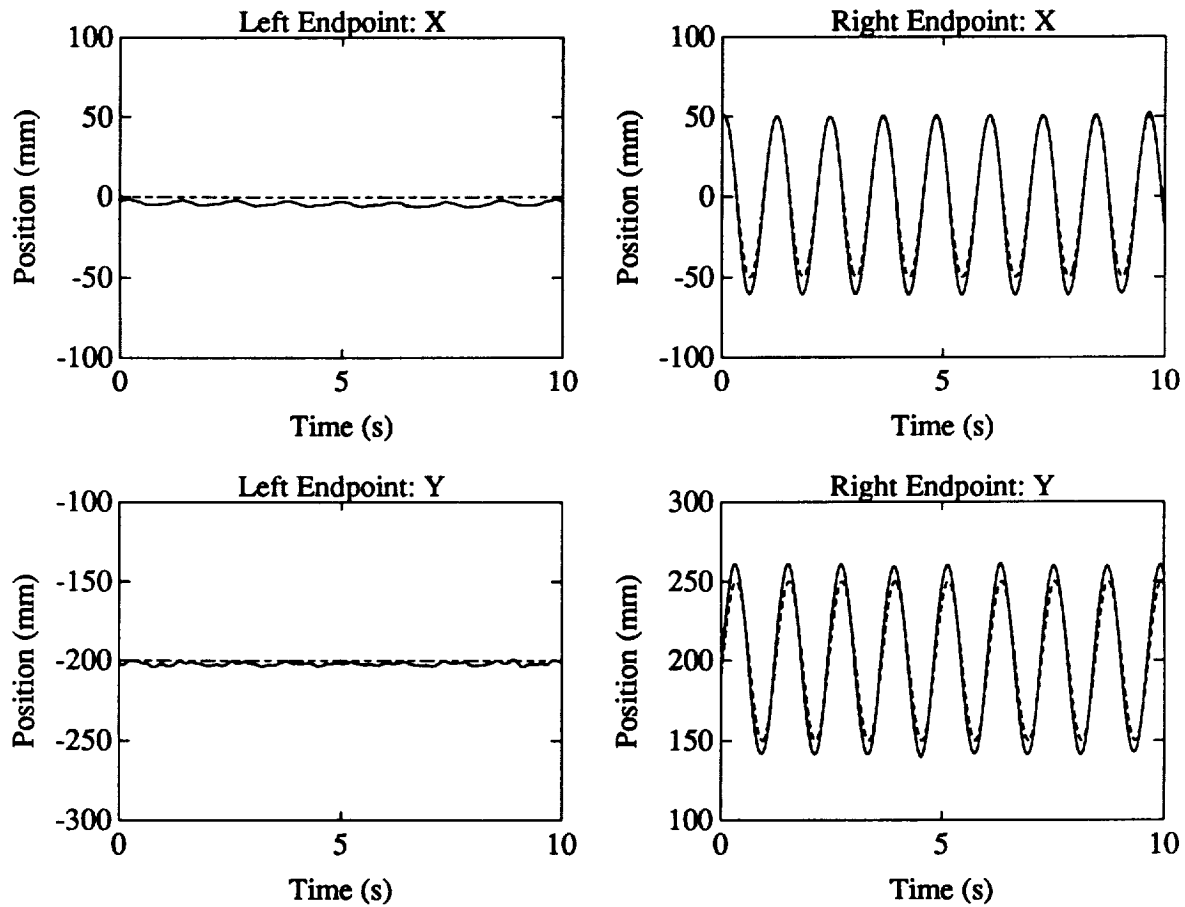


Figure 6.4: Two Arm Endpoint Position Control from a Free-Flying Robot

Experimental data illustrating simultaneous arm endpoint control from a free-flying robot. The left endpoint is controlled to a fixed location in inertial space, the right endpoint is controlled to follow a circular trajectory, both using endpoint feedback. The sinusoidal errors in the left endpoint position are primarily due to the unmodeled dynamics of arm joint spring forces (due to tubing and wiring), and joint friction forces.

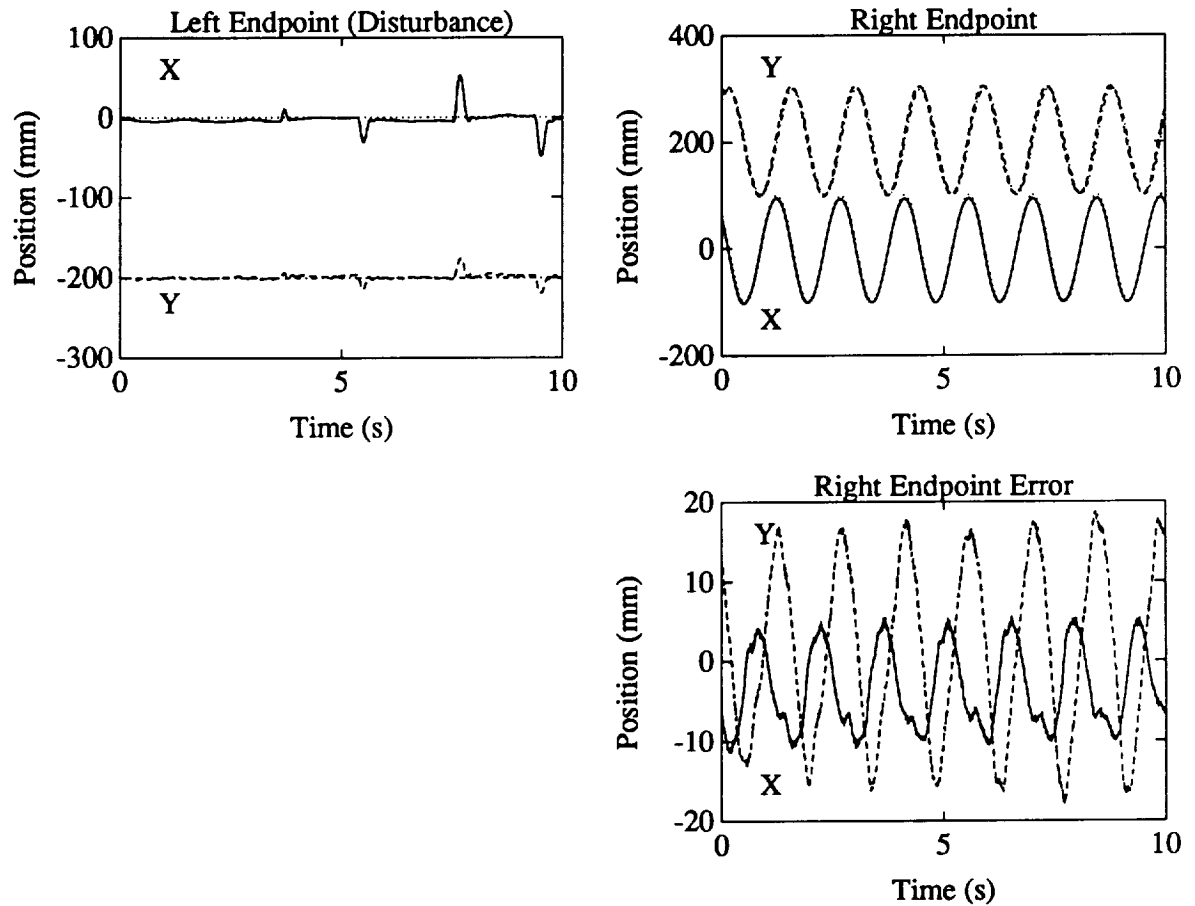


Figure 6.5: **Disturbance Rejection: One Arm Endpoint Disturbed on the Free-Flying Robot**

Notice how little cross-coupling between manipulator endpoint motion is visible in reaction to the disturbances.

In figure 6.5 disturbances introduced at one arm endpoint do little to affect the motions of the other. The left arm endpoint is disturbed from its controlled position, and recovers after the disturbance ceases. The response illustrates that there is little dynamic coupling between the two manipulator arms. The disturbance and recovery of the left endpoint has no noticeable effect on the right endpoint's positioning. The major reason for this is because the joints are free⁴, and motions introduced at one end of an arm do not introduce

⁴The joints have low friction and the low-friction motors have no gearing. The motors are torque supplying devices.

large forces at the other end. The articulated robot arm mechanisms act as vibration isolators between arm endpoints and bases.

Momentum was specified to remain constant to the controller in these experimental runs, so no thrusters or other momentum reaction devices were activated.

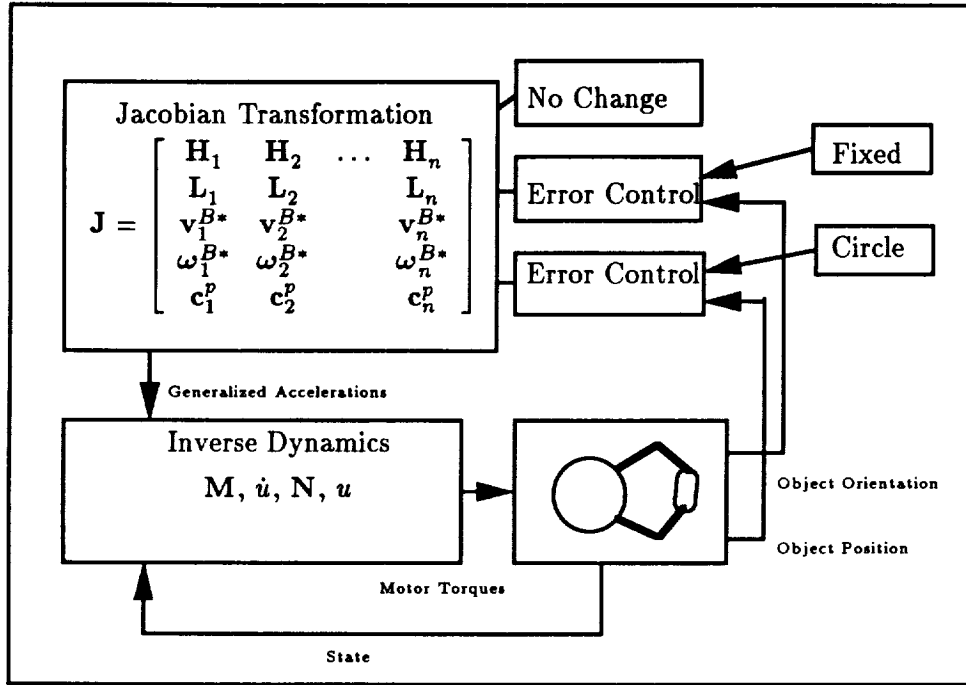


Figure 6.6: Overview of Computed-Torque Object Controller

6.2 Cooperative-Arm Object Manipulation

In this section, cooperative-arm control of an object from a free-flying robot using an automated CT controller is demonstrated. A full rigid-body dynamic model is used in the CT control system, a block diagram of which is presented in figure 6.6. The goal of this experiment is to show that the manipulated object can be precisely controlled in position and orientation, along with squeeze force⁵, in spite of robot base motion.

The orientation of the object is controlled to remain fixed in inertial space, while a point on the object, its centerpoint, will be controlled to follow a circular trajectory in

⁵The squeeze force between two manipulators is defined as the difference in force along their line-of-action.

inertial space. The motion of the object will cause the base to move in reaction. The force with which the manipulators squeeze the object will be controlled. The CT controller, which is told not to change system momentum, asks for no external forces or torques.

The components of the CT control system will be discussed first: the commanded trajectories, the dynamic system, the computed-torque controller and the error control laws. This is followed by a simulation run and the experimental results.

6.2.1 Object Centerpoint Trajectory

As listed in table 6.3, the object's orientation was commanded to remain fixed, and the object's centerpoint was commanded to follow a circular trajectory of radius 25 mm with a 2 second period (30 rpm). The trajectory generator for the circle provided desired position and velocity and also an estimated feedforward acceleration.

Position	x (mm)	y (mm)
Object Center	$50 \sin 2\pi \frac{50}{60} t$	$100 + 50 \cos 2\pi \frac{50}{60} t$
Orientation	θ (rad)	θ (deg)
Object Body	$\pi/2$	90

Table 6.3: Trajectory Specification for the Manipulated Object

The object's orientation is to be held fixed in inertial space, the object's centerpoint is to follow a circular trajectory in inertial space.

$$\mathbf{p}_{traj}^p = r \begin{bmatrix} \sin 2\pi\omega t \\ \cos 2\pi\omega t \end{bmatrix} + \mathbf{r}^{center} \quad (6.7)$$

$$\mathbf{v}_{traj}^p = 2\pi r \begin{bmatrix} \cos 2\pi\omega t \\ -\sin 2\pi\omega t \end{bmatrix} \quad (6.8)$$

$$\mathbf{a}_{traj}^p = -(2\pi)^2 r \begin{bmatrix} \sin 2\pi\omega t \\ \cos 2\pi\omega t \end{bmatrix} \quad (6.9)$$

6.2.2 Computed-Torque Controller

As in the previous experiment, the automated CT control program (RD) will be used for simulation and CT control of the two-armed free-flying robot. In this case, however, in

addition to the robot (consisting of bodies B, C, D, E, and F) there is an object G. The description of the dynamic system provided to RD is slightly different from that presented earlier in section 6.1.2 to account for this new object in the system:

```
%      Sample Configuration File for RD
%      Two-Armed, Free-Flying Robot with payload under Endpoint and Momentum Control
%      Dynamic System Description (all units in SI - kg, m, s)

DynamicSystem
(bodies B, C, D, E, and F as defined earlier)

Object      "G"      % Manipulated Object
AttachedAt   0.2953 0.0      % attachment point: gripper port 1
0.11 0.0      % center of mass location of object
1.2          % mass of object
0.008        % Izz inertia of object about cm
Constraint   "Right Endpoint"
0.22 0.0      % constrained endpoint: gripper port 2

Endpoint     "Object Centerpoint"
AttachedAt    0.11 0.0      % object's center
EndChain
EndDynamicSystem

% These quantities allow good estimates of object position and force
%
Inputs
    Position "Right Endpoint"
    Position "Left Endpoint"
    Force "Right Endpoint"
    Force "Left Endpoint"
EndInputs

% These quantities are of interest to the error controllers
%
Outputs
    Position      "Right Endpoint"
    Velocity      "Right Endpoint"
    Position      "Left Endpoint"
    Velocity      "Left Endpoint"
    Position      "Object Centerpoint"
    Velocity      "Object Centerpoint"
    SqueezeForce  "G"
EndOutputs
```

```

% These are the set of control objectives
%
Control
    LinearMomentum
    AngularMomentum
    Acceleration    "Object Centerpoint"
    AngAcceleration "G"
    SqueezeForce    "G"
EndControl

```

As before, the endpoint position determined by **RD** is used by the vision system to locate the endpoints in its field of view. Now, however, these measured endpoint positions are used by **RD** to infer the object's centerpoint position, much like a real free-flying robot would need to infer a position on a grasped body (such as a replacement module). This inferred position, along with the endpoint velocity determined by **RD** from base velocity and joint rates, is used in the error controller. It could also be possible to have the position and orientation of the object determined by a vision system, in which case these measured values could be used directly by the controller.

The complete set of control objectives for this system are as follows: system linear momentum, system angular momentum, object centerpoint "Object Centerpoint" acceleration, and object G angular acceleration. The augmented Jacobian technique discussed in chapter 3 is used. Since this is a closed-chain mechanism, a motion constraint is implicitly included in the control objectives. The zero value for the constraint objective was discussed in section 3.3.3. The (dynamic) control objectives are stated here:

$$\mathbf{a}^S \stackrel{\triangle}{=} \underset{3.29}{\begin{bmatrix} \frac{d}{dt} \mathbf{L}^S \\ \frac{d}{dt} \mathbf{H}^{S/S^*} \\ \mathbf{a}^{\text{ObjectCenterpoint}} \\ \boldsymbol{\alpha}^G \\ 0 \end{bmatrix}} \quad (6.10)$$

The squeeze force, while controlled in this system, is not a dynamic quantity⁶: it is purely a function of the kinematics and the applied joint torques. The system's kinematics (joint

⁶The squeeze force can be determined using the principles of statics – dynamics are not required.

angles) can be used to determine joint torques that correspond to a squeeze force independently of the dynamic quantities. Of course, an error controller on the squeeze force can use measured endpoint forces to compensate for errors in the sensed joint angles and errors in the actual torque applied by the actuators on the manipulators.

RD will automatically evaluate the Jacobian equation and inverse dynamics equations to solve for manipulator torques given these control objectives. It can also, at program command, evaluate and solve the system dynamical equations of motion for simulation.

6.2.3 Error Controllers

The error controllers ensure that the object's centerpoint and orientation track commanded trajectories, and that the object squeeze force is regulated. The control objectives for this experiment are the angular acceleration of the object orientation, the acceleration of the object's center-point, the squeeze force on the object, and the system momentum rates.

The error controllers for endpoint positions determine desired endpoint accelerations to cause the error to behave in a known manner. As discussed in the previous experiment, feedback control is necessary to reduce the errors caused by unmodeled dynamics (wiring spring and friction forces in the arm joints). In the case of the second-order position controllers, this response was a damped sinusoid with the same bandwidth response as previously.

The object's squeeze force f is controlled by a feedforward component $\mathbf{F}_{desired}$ and integral error feedback⁷. The center of the payload p is position controlled, and the orientation of the payload body G is orientation controlled, so the control objectives in response to object trajectory errors and object squeeze force errors are:

$$\mathbf{a}_{commanded}^p = K_p(\mathbf{p}_{des}^p - \mathbf{p}^p) + K_v(\mathbf{v}_{des}^p - \mathbf{v}^p) \quad (6.11)$$

$$\boldsymbol{\alpha}_{commanded}^B = K_p^\alpha(\theta_{des}^G - \theta^G) + K_v^\alpha(\dot{\theta}_{des}^G - \dot{\theta}^G) \quad (6.12)$$

$$\mathbf{F}_{commanded} = K_i \int_0^t (\mathbf{F}_{measured} - \mathbf{F}_{desired}) + \mathbf{F}_{desired} \quad (6.13)$$

The values for the position gains K_p and velocity gains K_v are listed in table 6.4. The position and velocity gains for position control, as in the previous experiment, were chosen

⁷No proportional feedback was used: proportional feedback in a system with no dynamics causes problems when run in discrete time.

for the desired bandwidth of response. The effect of the sensor noise on endpoint control becomes more evident as the payload mass and inertia increase: computed-torque control scales the manipulator torques to the mass parameters of the manipulated object.

	$K_p \text{ (m/s}^2\text{)/(m)}$	$K_v \text{ (m/s}^2\text{)/(m/s)}$
Object Centerpoint	100	22
	$K_p^\alpha \text{ (rad/s}^2\text{)/(rad)}$	$K_v^\alpha \text{ (rad/s}^2\text{)/(rad/s)}$
Object Orientation	100	22
	$K_i \text{ (N)/(N-s)}$	
Squeeze Force	0.2	

Table 6.4: **Error Controller Gains for Position and Orientation and Squeeze Force**

The position controller uses a second-order control law. The force controller uses an integral control law.

The force control integral feedback gain was chosen to provide a slow response to errors. Motion of the endpoint gripper mechanisms within the gripper ports⁸ cause sudden spikes on the force sensors and can affect the integrated error sufficiently to saturate the arm actuators if the feedback gain K_i is high. Saturating the arm actuators typically results in sudden odd motions. When the force integral-feedback gain is zero this effect does not occur.

6.2.4 Simulation Run

An animation of this system with its controller in operation is presented in figure 6.7⁹, which shows the object behaving correctly: it has constant orientation, and the center is following a circular trajectory about (0,0). The robot body, although initially at rest, moves in response to manipulator activity. The majority of the base motion in response to manipulator activity is angular, but there is also noticeable translational motion. The mass of this object (6.9 kg) causes the base reactions to be somewhat larger than those without an object, shown in figure 6.3. The controller and simulation dynamic models

⁸The fit of the grippers in the ports is not very tight, and the gripper can move around a bit (estimated at 0.5 mm).

⁹The matlab script file for this simulation is presented in appendix D.

were the same for this simulation run. Actual robot coordinate information was used as an initial condition for this simulation: automatic numerical relaxation of the constraints is performed by the simulation code using the augmented constrained dynamics equations developed in section 2.4.2.

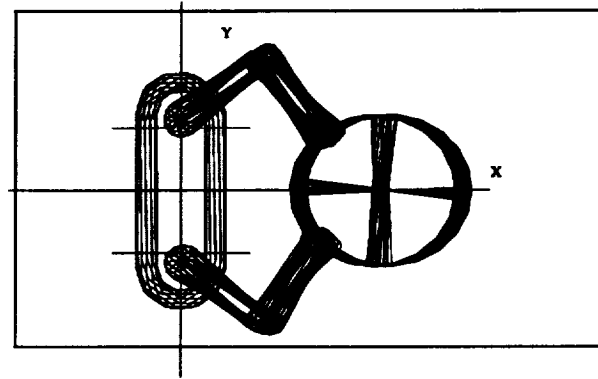


Figure 6.7: **Animated Matlab Simulation Run of Object Control from a Free-Flying Robot**

This is a dynamic simulation of the experimental robot manipulating a 6.9 kg object under endpoint control. Notice that the resulting robot base motion is primarily angular in nature. (Robot base translation is of course larger than that seen without a payload, as in figure 6.3.)

6.2.5 Experimental Results

The results of three experimental runs are presented: one that shows the object following its commanded trajectory in space, one that shows the response of the squeeze force controller, and one that shows disturbance rejection. The controller's sample rate is 60 Hz¹⁰. Object position and orientation were inferred from the manipulator endpoint positions, which were obtained directly from the vision system, object velocity and angular velocity were estimated given the base velocity (from the vision system base-position measurements), the base angular-speed measurement from the angular-rate sensor, and manipulator-arm joint rates. Velocities determined this way had substantially lower noise than velocities obtained from differencing endpoint position information, due to quantization of the vision

¹⁰Due to the vision system frame rate

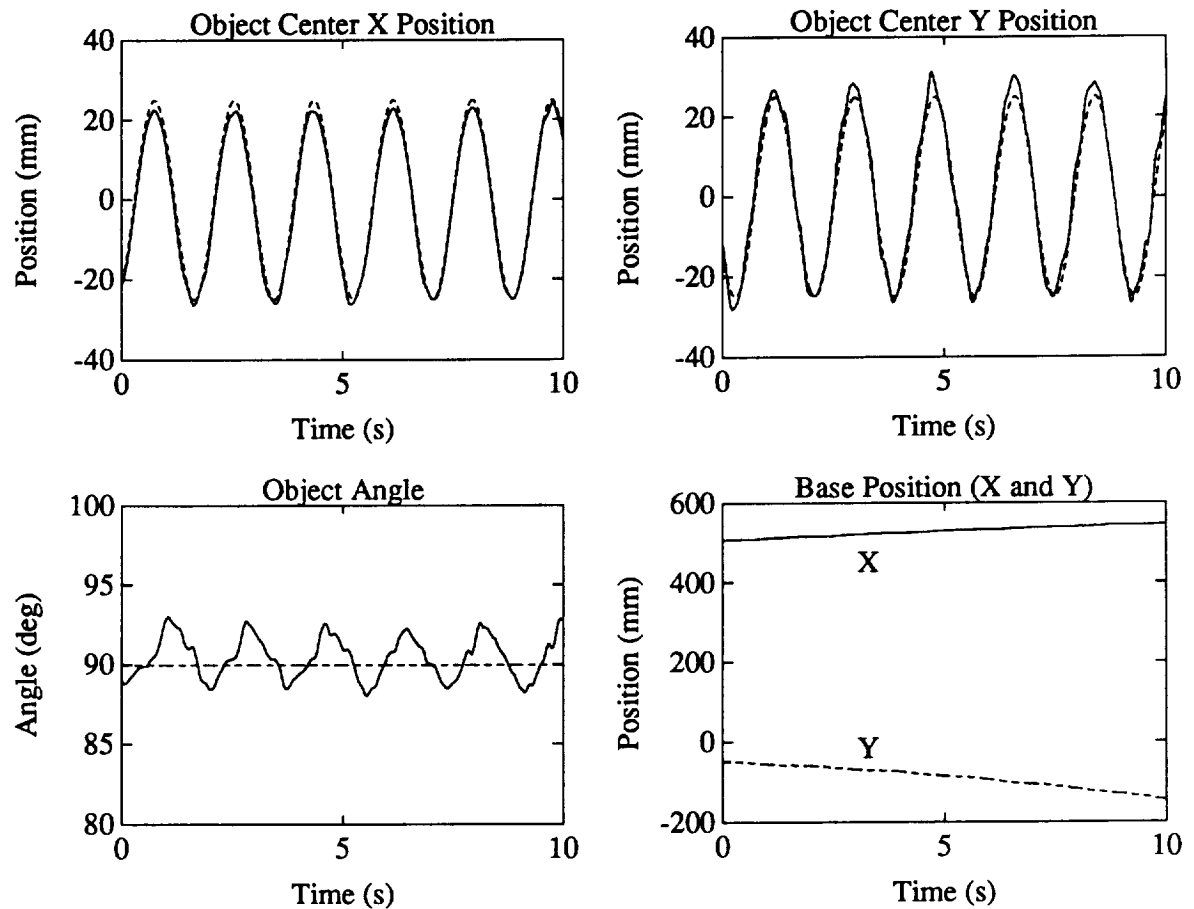


Figure 6.8: **Cooperative-Arm Object Position Control from a Free-Flying Robot**

Experimental data illustrating cooperating-arm object position control from a free-flying robot. The object's orientation is controlled to a fixed orientation in inertial space, the object's centerpoint is controlled to follow a circular trajectory about (0,0). For these three subplots, the solid lines are experimental data, the dashed lines are the desired response. The sinusoidal errors in the object orientation are primarily due to the unmodeled dynamics of arm joint spring forces (due to tubing and wiring), and joint friction forces. A plot of base motion shows its initial translational motion only slightly affected by object motions.

sensor. This controller consumed 75% of the CPU time on the 25 MHz 68030 real-time microcomputer.

The performance of the object's centerpoint position controller is shown in figure 6.8. The object orientation is held near 90 deg, as commanded, even though the robot has an appreciable translational velocity. As with the endpoint controller in the previous experiment, unmodeled torques from the wiring and tubing in the manipulator arms cause errors. These errors can be difficult to suppress in a feedback control system when the sensors are noisy: turning up feedback gains results in large actuator forces due to noise amplified in the feedback system. The angular rate of the object is inferred from the base angular rate and the manipulators' angular rates. This kinematically derived estimate, although less noisy than pseudo-differentiating vision system information, is still subject to the noisy information from the base angular rate sensor.

The robot base's motion is primarily angular in reaction to moving the object. The reactions when moving an object are of course larger than when moving just the manipulators. Moving a heavy object (6.9 kg) side-to-side more than a few centimeters causes the robot to rotate enough to exceed the range of motion of its manipulator arms. If heavy objects such as this are to be successfully manipulated over larger distances, some form of momentum control will need to be employed by the robot. It is important to note that *angular* motion – unlike linear motion – is easy to control without mass expulsion. Chapter 8 discusses future research to investigate this problem.

Experiments showing the performance of the squeeze force controller in response to a step command, with and without error feedback, are shown in figure 6.9: the unmodeled dynamics that affect position control are not as significant in the force control. The squeeze force *by definition* is non-dynamic in a purely rigid body system,¹¹ so barring miscalibrations, no changes in object position or orientation are expected. In practice, too, changing the squeeze force results in very little motion of the object. Furthermore, because this cooperative-arm control technique does not resolve motion constraints with position, the object squeeze force is not affected by errors in joint angle measurements. This is evident in the accuracy of the force control achievable experimentally without *any* feedback.

¹¹In a rigid body system, the force can be controlled purely by changing actuator forces: no motion results.

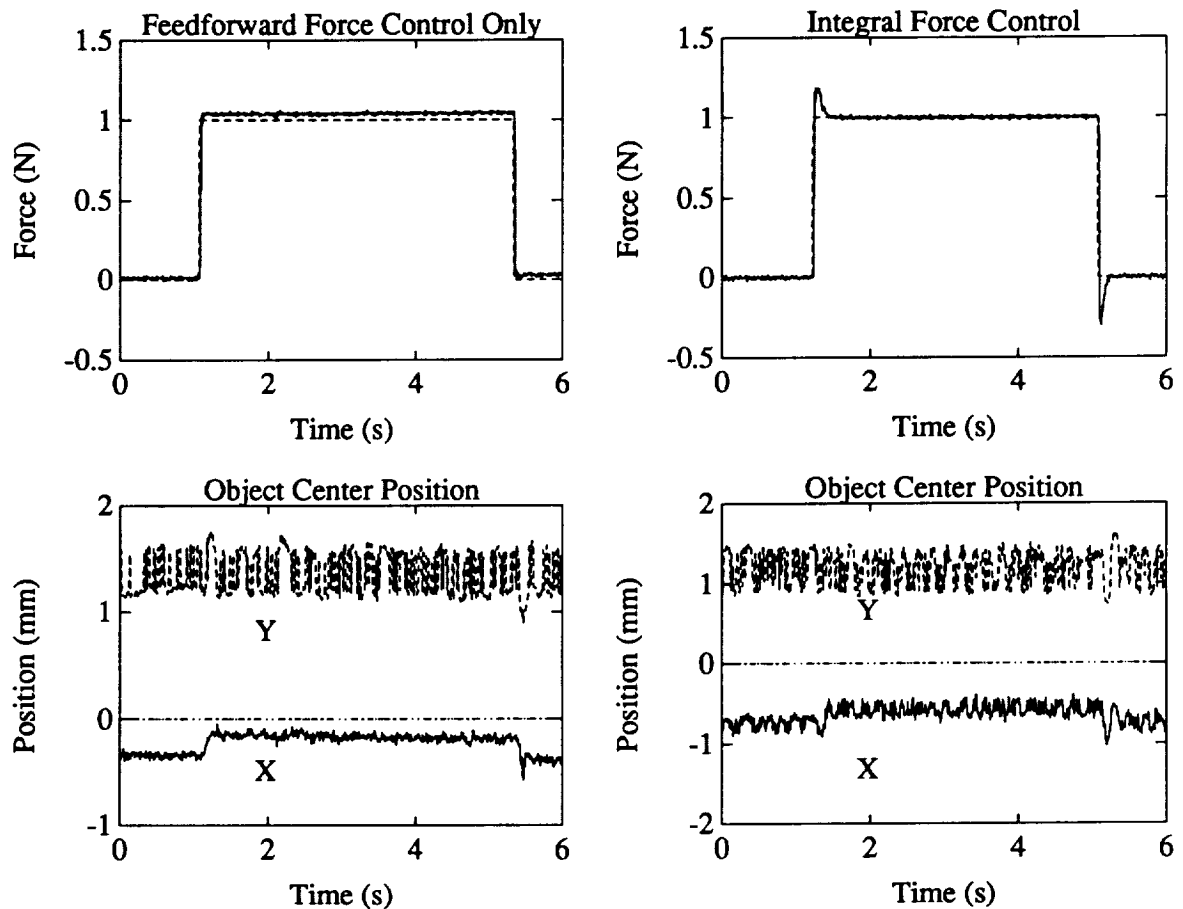


Figure 6.9: Cooperative-Arm Object Squeeze-Force Control from a Free-Flying Robot

Experimental data illustrating object squeeze-force control from a free-flying robot. On the left side, only feedforward forces are used, on the right side, feedforward with integral feedback control is used. With pure feedforward, force control within 10% is achieved: the motor torques and system angle sensors are well calibrated. When integral force feedback control is used the steady state error is near zero, but an overshoot occurs as the integrator adapts to a new offset. Note how little the object moves in response to changed force commands – motion is almost in the sensor noise. Ideally, it would not move at all.

The small errors experienced can be attributed to unmodeled dynamics such as friction in manipulator bearings and spring forces introduced by wiring and pneumatic tubes in the arms, and mismodeling of the sensors and actuators¹².

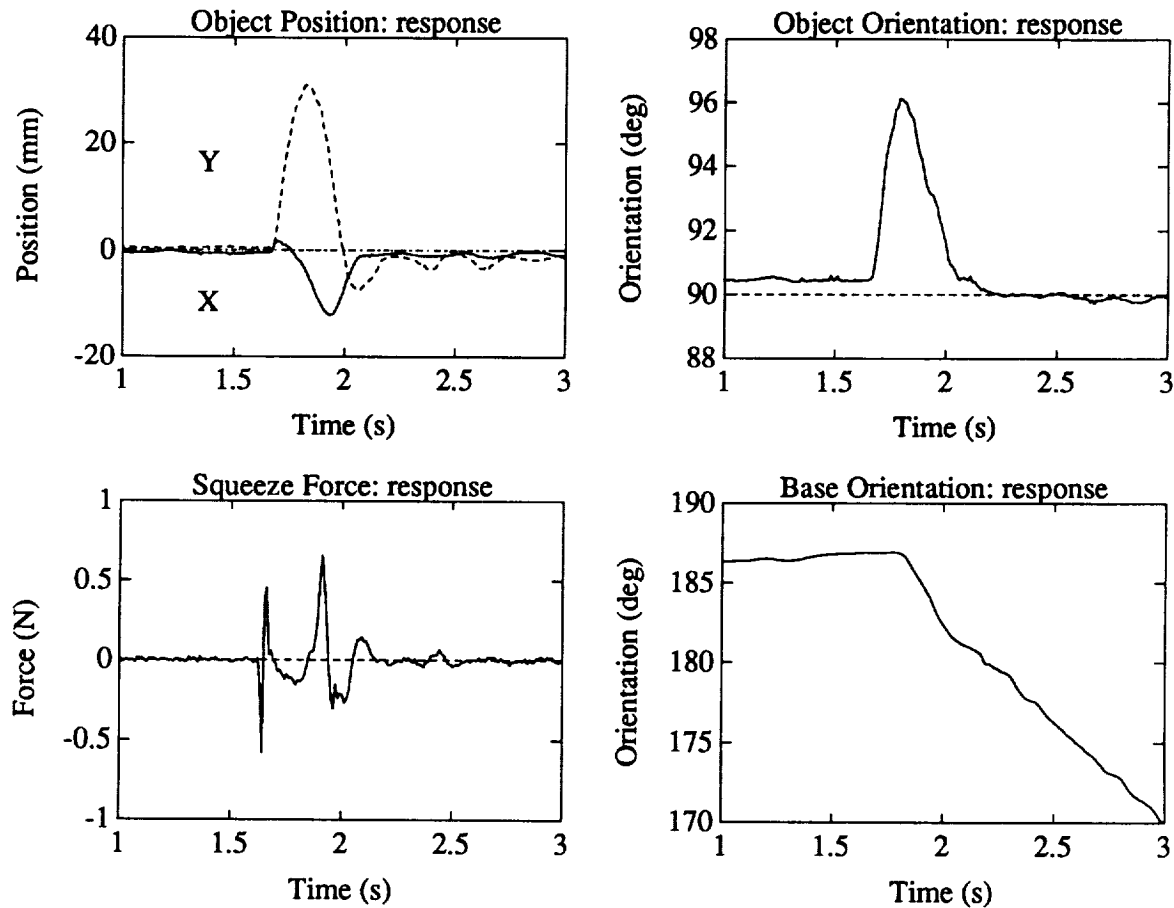


Figure 6.10: **Cooperative-Arm Object Control from a Free-Flying Robot: External Disturbance Rejection**

The object is disturbed briefly by an external force. Small spikes in the squeeze force measurement are due to the mechanical play in the object gripper mechanism. The impulse from the external disturbance is rejected by the object controller: the object quickly returns to the desired position and orientation. The impulse does, however, affect the momentum of the system: it sets the robot base into rotation.

The response to disturbances of object positioning and squeeze force are shown in

¹²Gain errors, time-varying biases and sensor nonlinearities exist.

figure 6.10. The object is briefly tapped: the object position and orientation are quickly returned to the desired value. The squeeze force undergoes some small spikes due to mechanical play in the gripper mechanism; however, these are very small, approximately 0.5 N. The impulse from the disturbance changes the system momentum, and since the object is controlled to be stationary, the robot base's velocity and angular velocity must change.

Momentum was specified to remain constant (to the controller), so no thrusters or other momentum reaction devices were activated.

6.3 Summary

In this chapter, the automated CT control computer program (**RD**) developed in chapter 4 was used to do dynamic simulations of the experimental robot system, and CT control of the experimental robot itself.

Dynamic simulations of the two-armed free-flying robot system using the **RD** augmented simulation equations predicted base motions in response to manipulator motions, and increased base motions as a heavier payload were moved. The ability of these simulation equations to numerically relax the closed-chain constraints allowed full-state information read out of the experimental robot to be plugged into the simulation directly, even although it was not precisely consistent.

Simultaneous two-arm endpoint control experiments demonstrated the ability to control both arm endpoints by using an endpoint sensor and a feedback control system based on the full free-flying multiple rigid-body system. Cooperative-arm object manipulation experiments demonstrated the ability to control object position, orientation and squeeze force: important when manipulating fragile objects. Overall, controller performance was quite satisfactory in the presence of the sensor noise (body angular rate sensor, vision sensor) and sample-rate limitations.

Chapter 7

Can Base Accelerations be Neglected?

In the previous chapter, endpoint and object control from a free-flying robot were demonstrated using a full free-flying rigid-body model in the CT controller. Modeling full dynamics in the CT controller, while “exact” in solution, is computationally very expensive. For spacecraft, much greater numeric processing requirements mean much greater computer complexity, lower reliability, greater power requirements, and increased vehicle mass. If a control algorithm is to be viable for space-borne robots of the near future, it must be as simple as possible while meeting performance requirements.

The conditions under which a complex model in the CT controller is necessary, and under what conditions a simplified model will perform adequately are examined in this chapter. In this chapter, two simplifications are examined: neglecting base *velocities* and neglecting base *accelerations*. It is found that neglecting the base angular velocity prevents accurate compensation for nonlinearities, and saves little computation; but neglecting base *accelerations* is quite tolerable and can save a significant amount of computation. Mismodeling due to neglecting base accelerations is compared to that due to uncertainties in mass distribution: errors in mass distribution of 5% can result in acceleration errors of 5% when using the CT control method.

Specifically, the effect of neglecting base accelerations is predicted and experimentally verified for two test cases with the experimental robot. These two experiments suggest

that, for the robot mass parameters and manipulator articulation used in the experiments, neglecting the base *accelerations*¹ can result in negligible endpoint controller error. Simulation results then show many other cases in which base accelerations could be neglected, and others where full modeling may be desirable. On the basis of this combined study a wide range of free-flying robot system parameters is established wherein it is possible to use a simplified endpoint CT controller: one that neglects robot base accelerations. It is shown that a tool such as **RD** can be used to predict controller performance and determine that a particular design fits within this regime.

A consequence of neglecting robot base acceleration in the simplified CT controller is that the Jacobian augmentation terms for momentum control or base control are no longer included: this control will then need to be external to the manipulator controller.

7.1 Simplified Computed-Torque Endpoint Control

Consider, for example, the case where a robot is able to impart forces or torques via massless (e.g., magnetic) actuators to a payload. The payload can then be made to accelerate as desired. The robot's mass and inertia characteristics will affect the way the robot reacts, but *not* the way the payload accelerates. It would then be possible to ignore all aspects of the free-flying nature of the robot in the manipulator controller's model; to assume it fixed².

Real robots, however, have non-zero mass and inertia in their manipulators themselves, and using a fixed-base model in the CT controller can adversely affect performance. Although changes in configuration of the manipulator are compensated for by the CT controller, performance problems will come from unmodeled inertial forces and torques due to the linear and angular acceleration and angular rotation of the base. These two components have different impacts on the performance of the control system, and have radically different computational costs.

Previously published experimental work in endpoint control from free-flying robots by Alexander [1], Umetani and Yoshida [36], Carignan [3], and theoretical work by Vafa

¹The accelerations considered are the linear accelerations of the center-of-mass of the robot body, and the angular accelerations of the robot body.

²Station-keeping control would still be required, of course.

and Dubowski [7], has concentrated on formulating new controllers to compensate for all of the dynamics of a free-flying system. Masutani et al [41] were the only ones truly to investigate (in simulation) the performance of simplified endpoint controllers on free-flying manipulators. They used Umetani and Yoshida's generalized Jacobian technique to do Jacobian transpose control. This method, unfortunately, ignores all dynamics in the system.

Up until very recently, there has been no investigation of the use of simplified dynamic models for endpoint CT controllers on free-flying robots. In newly published work, Papadopolous and Dubowski [22] suggest that "*nearly any* control algorithm that can be used for fixed-base manipulators can be also employed in the control of free-flying systems, based on the structural similarities between the kinematic and dynamic equations of a free-floating space system and the equations for the same manipulator with a fixed-base".

Fixed-base robot manipulator endpoint CT control laws are in fact simplified endpoint CT controllers. CT control, by definition, already handles changes in manipulator configuration, a feature of moving free-flying robots. The real differences that separate full free-flying CT controller models from fixed-base CT controller models are the unmodeled inertial forces and torques due to the angular velocity and the linear and angular acceleration of the base of the free-flying robot. These components have different impacts on the performance of the control system depending on the articulation and mass distribution of the robot/payload system, and will be examined separately.

7.1.1 Neglecting Robot Base Angular Velocity

The earth is massive, and its angular velocity is small: its reference frame approximates a secondary Newtonian frame for many applications, and fixed-to-earth robotic manipulation may be performed in this slowly rotating reference frame using a fixed-base model controller.

If, however, the angular velocity of a robot manipulator's base is large (as it can be when a robot is free-flying), then its reference frame is non-Newtonian. Points along the manipulators will experience centripetal accelerations due to the robot's angular velocity. In the full-model CT control scheme, these additional centripetal accelerations are

accounted for in the inverse dynamics process – the least computationally expensive part of CT control. In the **RD** program, the cost of including centripetal acceleration terms in the kinematics and inverse dynamics is 47 total additional operations³ – equivalent to the amount of computations required in the recursive algorithm for one additional link. This computational cost is low because of the recursive algorithm used to compute the kinematics and inverse dynamics: the compensation is introduced at the start of the kinematic chains and is then automatically propagated.

For example, a 2D free-flying robot with two arms requires 142 operations to evaluate the forward kinematics and 123 operations to evaluate the inverse dynamics. Including the base angular speed adds 47 computations, 20%, to the cost of this portion of CT control. This additional cost is small compared to the cost of solving the system Jacobian, however. This will be examined next.

7.1.2 Neglecting Robot Base Accelerations

Free-flying robot base linear and angular *accelerations* occur in response to manipulator activity and external forces. If the robot body's mass and inertia are very large it is inconsequential to assume zero robot base accelerations; for the robot base accelerations will be small compared to manipulator endpoint accelerations and manipulator articulation will usually isolate the manipulator endpoint from base linear and angular accelerations. If the robot's base is not very large, this will not be true.

Unfortunately, unlike for base angular velocity (discussed previously), including base *accelerations* in a CT model is *not* cheap.⁴ In a full-model CT controller, the system's accelerations, including base accelerations, are solved for by inverting the Jacobian matrix. The accelerations of the robot base can be solved along with the manipulator accelerations if the manipulator Jacobian is augmented with momentum terms. If the base accelerations are assumed to be zero (or non-varying), then a Jacobian suitable for a fixed-base robot⁵ can be used to solve for the manipulator generalized accelerations. Since the solution of

³See table 4.1

⁴Constant base acceleration, such as that due to rocket (thruster) activity can be easily compensated for: here we consider base accelerations in response to manipulator activity.

⁵Not containing the momentum augmentation terms.

the Jacobian equation requires $(\frac{2}{3}n^3 + 2n^2 - \frac{2}{3}n)$ operations⁶, the reduction of n by 3 (for 2D systems) and 6 (for 3D systems) can produce substantial computational savings. For example, a 3D robot with a single 7 degree-of-freedom manipulator arm would require 1794 operations to solve the full Jacobian inverse⁷, and 331 operations to solve the simplified Jacobian inverse. In fact, greater savings in computation than this are possible if the robot has multiple manipulators: an example follows.

7.1.3 Neglecting Base Accelerations: An Example

Solving a simplified Jacobian matrix equation, as opposed to a full model Jacobian matrix equation, offers substantial computational savings. For example, a two-arm endpoint controller Jacobian for a free-flying 3D robot has the form:

$$\mathcal{J} = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_2 & \dots & \mathbf{H}_n \\ \mathbf{L}_1 & \mathbf{L}_2 & & \mathbf{L}_n \\ \mathbf{v}_1^{p^1} & \mathbf{v}_2^{p^1} & & \mathbf{v}_n^{p^1} \\ \mathbf{v}_1^{p^2} & \mathbf{v}_2^{p^2} & & \mathbf{v}_n^{p^2} \end{bmatrix}_{n \times n} \quad (7.1)$$

The momentum terms augment the normal manipulator Jacobian to allow the solution for base as well as manipulator accelerations. The order of the system, if the two arms each have 7 degrees of freedom is $n = 6 + 7 + 7 = 20$. The basic manipulator Jacobian, not including momentum terms, for this two-arm 3D robot is:

$$\mathbf{J} = \begin{bmatrix} \mathbf{v}_1^{p^1} & \mathbf{v}_2^{p^1} & \dots & \mathbf{v}_n^{p^1} \\ \mathbf{v}_1^{p^2} & \mathbf{v}_2^{p^2} & & \mathbf{v}_n^{p^2} \end{bmatrix}_{(n-6) \times (n-6)} \quad (7.2)$$

The manipulator accelerations in this Jacobian are independent for solution purposes – to be expected in a fixed-base system. The two manipulators (and their Jacobian matrices) can therefore be treated independently.⁸ The Jacobian can then be chopped into two, one small Jacobian for each manipulator, each of which looks like:

$$\mathbf{J}^i = \begin{bmatrix} \mathbf{v}_1^{p^i} & \mathbf{v}_2^{p^i} & \dots & \mathbf{v}_n^{p^i} \end{bmatrix}_{\frac{(n-6)}{2} \times \frac{(n-6)}{2}} \quad (7.3)$$

⁶See table 4.1

⁷Actually, Solving $Ax = b$, for x .

⁸This will work if the generalized speeds are defined so that they do not mix joint angle rates of the two manipulators. A choice of $u_i \triangleq \dot{q}_i$ guarantees this.

Neglecting base accelerations and solving two Jacobian equations of half the size $\frac{(n-6)}{2}$ can yield quite a computational saving. Continuing the example presented above, a 3D robot with two 7 degree-of-freedom manipulator arms ($n=20$) would require 6132⁹ operations to solve the full Jacobian equation, 2212 operations after neglecting base accelerations, and only 662 operations after the manipulator Jacobian matrix is split.

This brief example serves to illustrate that substantial computations – approximately an order of magnitude – can be saved if base accelerations of a free-flying robot's base are neglected in the CT control system. The example also shows one of the fundamental difficult aspects of the CT control technique: the necessity of solving a matrix equation. It makes the cost prohibitive of using the CT control technique for systems in which many degrees of freedom must be modeled.

7.2 Simplified Modeling

The two differences between a fixed-base and free-flying CT controller model – base velocities and base accelerations – affect how both the Jacobian equation solution and the inverse dynamics are done. The simplified controller model that will be studied here *neglects* base accelerations but *includes* modeling of the base angular velocity. The simplified Jacobian equations obtained by neglecting base accelerations are *not* identical to fixed-base equations if robot base angular velocity is to be accounted for, although the simplified Jacobian is a fixed-base manipulator Jacobian. The base angular velocity enters into the model as nonlinear terms in the Jacobian equation, and as nonlinear terms in the inverse dynamics.

The equations used for CT control to deliver the joint torques, given desired endpoint accelerations, are similar to the full free-flying model equations, except for the Jacobian equation. The Jacobian equation will be used to solve for a subset of the system generalized accelerations; the manipulator generalized accelerations. The base accelerations will be assumed to be zero. The effect of this is to remove the rows of the Jacobian equation dealing with momentum control or base acceleration control (it depends on what the control objective may have been), and removing the columns of the Jacobian matrix that depended on the base accelerations. The resultant Jacobian is the same as the manipulator Jacobian

⁹See table 4.1

would be if the manipulator were fixed-base.

An example involving a two-link arm on a 2D free-flying robot will be used to illustrate this simplification. This system has 5 degrees of freedom: 3 of the base and 2 of the manipulator. The full-model control objectives include the system's linear momentum and angular momentum about the system's mass center and the acceleration of the manipulator endpoint p . The augmented Jacobian matrix to solve for the accelerations exactly is

$$\mathbf{J}_{3.35} = \begin{bmatrix} \mathbf{L}_1^S & \mathbf{L}_2^S & \mathbf{L}_3^S & \mathbf{L}_4^S & \mathbf{L}_5^S \\ \mathbf{H}_1^{S/S^*} & \mathbf{H}_2^{S/S^*} & \mathbf{H}_3^{S/S^*} & \mathbf{H}_4^{S/S^*} & \mathbf{H}_5^{S/S^*} \\ \mathbf{v}_1^p & \mathbf{v}_2^p & \mathbf{v}_3^p & \mathbf{v}_4^p & \mathbf{v}_5^p \end{bmatrix}_{5 \times 5} \quad (7.4)$$

The approximate system accelerations, assuming zero base accelerations, are

$$\dot{\mathbf{u}}_{approx} \stackrel{7.3}{=} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \dot{\mathbf{u}}_4 \\ \dot{\mathbf{u}}_5 \end{bmatrix}_{5 \times 1} \quad (7.5)$$

If this simplification is applied, the manipulator Jacobian matrix becomes

$$\mathbf{J}_{7.3,7.5} = \begin{bmatrix} \mathbf{v}_4^p & \mathbf{v}_5^p \end{bmatrix}_{2 \times 2} \quad (7.6)$$

and the row-reduced augmented Jacobian derivative matrix $\dot{\mathcal{J}}^*$ is

$$\dot{\mathcal{J}}^* \stackrel{7.4}{=} \begin{bmatrix} \dot{\mathbf{v}}_1^p & \dot{\mathbf{v}}_2^p & \dot{\mathbf{v}}_3^p & \dot{\mathbf{v}}_4^p & \dot{\mathbf{v}}_5^p \end{bmatrix}_{2 \times 5} \quad (7.7)$$

It is not the same as that for a fixed-base robot: the base angular velocity terms $\dot{\mathbf{v}}_3^p$ are included, and the base linear velocity terms $\dot{\mathbf{v}}_1^p$ and $\dot{\mathbf{v}}_2^p$ are present but zero (as always).

The generalized accelerations for the manipulator can each be determined with the following simplified Jacobian equation:

$$\dot{\mathbf{u}}_{4,5,7.4,7.6} = (\mathbf{J})^{-1}(-\dot{\mathcal{J}}^* \mathbf{u} + \mathbf{a}_{des}^p) \quad (7.8)$$

The two-arm generalized accelerations can be combined with the zero base acceleration to yield an approximate system generalized acceleration, as shown in equation 7.5.

The inverse dynamics process will use the approximate acceleration solved for in equation 7.8 with the *full* system state to determine actuator torques. The inverse dynamics equation must be done with the full free-flying equations of motion:

$$\begin{aligned}\mathbf{F} &= \mathbf{M}\dot{\mathbf{u}}_{approx} + \mathbf{N}\mathbf{u} \\ \boldsymbol{\tau} &= \mathbf{W}^{-1}\mathbf{F}\end{aligned}$$

where the matrix \mathbf{W} maps generalized speeds into the derivatives of the generalized coordinates (joint rates).

$$\dot{\mathbf{q}}_r = \sum_{s=1}^n \mathbf{W}_{rs} \mathbf{u}_s$$

For CT control, a recursive Newton-Euler algorithm discussed in section 4.4.2 is preferred: it yields results identical to those produced by the use of this equation.

By using a simplified CT controller such as this, the ability to control momentum is lost: it will need to be done by an external controller. In the event that momentum control (for station-keeping) produces large base accelerations $\dot{\mathbf{u}}_{1..3}$, the Jacobian equation can include them, but as constants. These constants can be included using a partition of the augmented Jacobian $\mathcal{J}_{1..3,4..5}$: those columns that were neglected can now be included, but are treated as invariant to manipulator accelerations.

$$\dot{\mathbf{u}}_{4,5,7,4,7,6} = (\mathbf{J})^{-1}(-\dot{\mathcal{J}}^* \mathbf{u} + \mathbf{a}_{des}^p - \mathcal{J}_{1..3,4..5} \dot{\mathbf{u}}_{1..3}) \quad (7.9)$$

In summary: a simplified free-flying manipulator Jacobian matrix equation (one of 7.8 or 7.9) has many similarities to the fixed-base manipulator Jacobian matrix equation, but is not identical to it. The order of the matrix equation is reduced by the number of degrees of freedom due to free-flight (3 in 2D, 6 in 3D). This method allows smaller Jacobian matrices to be used (with corresponding savings in computation), while providing accurate compensation for all the nonlinear terms: nonlinear terms are a function of the state, not of the base accelerations.

7.3 Errors due to Mismatching

Over what range of free-flying robot parameters and/or payloads is neglecting base accelerations permissible? Unmodeled or mismatched dynamics in a CT controller are responsible

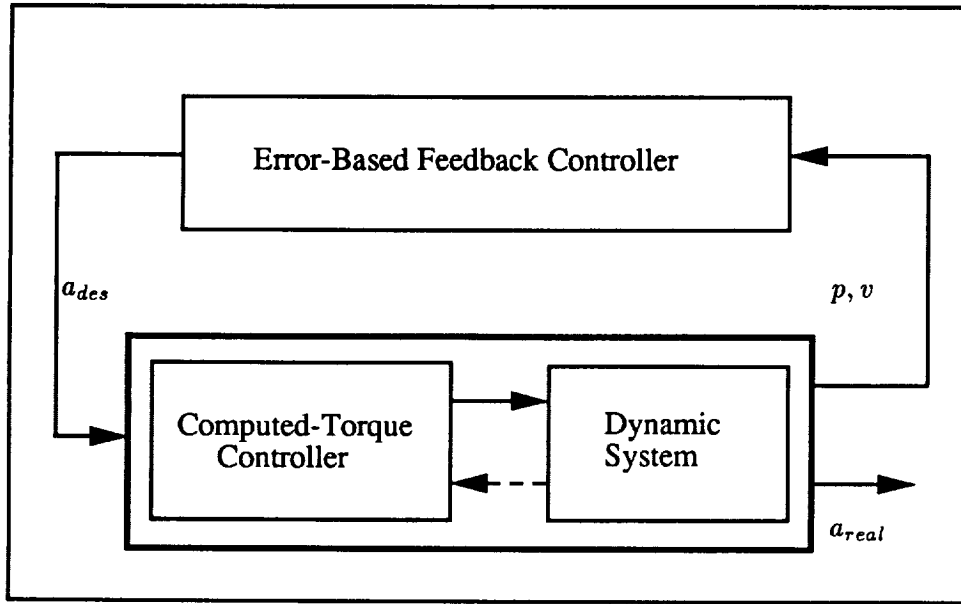


Figure 7.1: **An Error-Based Feedback Controller**

Errors between desired accelerations and actual accelerations eventually show up in measured positions and velocities. The feedback controller acts on position and velocity errors.

for a mismatch between accelerations in the real system and accelerations asked by the control system. A typical feedback control system schematic is shown in figure 7.1. One measure of the accuracy of a computed-torque endpoint controller is how well the physical system matches the desired accelerations asked of the controller. In general, there will be imperfections due to friction, imperfect sensor measurements, and imperfect knowledge of robot mass distribution. If the dynamic system is modeled with simplifying assumptions, then there will also be errors due to the unmodeled dynamics. These model errors result in incorrectly computed torques, and hence incorrect system endpoint accelerations, for which feedback must compensate, albeit imperfectly.

These incorrect endpoint accelerations will be the sum of (a) a linear function of the desired accelerations and (b) unmodeled nonlinear components:

$$\mathbf{a}_{\text{endpoint}} = \begin{bmatrix} T_{xx} & T_{xy} \\ T_{yx} & T_{yy} \end{bmatrix} \mathbf{a}^{\text{desired}} + f_{\text{NL}} \quad (7.10)$$

$$= \mathbf{T} \mathbf{a}^{\text{desired}} + f_{\text{NL}} \quad (7.11)$$

The transformation matrix \mathbf{T} describes endpoint accelerations seen at the manipulator tip in response to accelerations asked of the control system when the simplification of zero base accelerations is made. Ideally, this matrix is unity, and the nonlinear terms f_{NL} are zero.

Data plots will be presented that contain the values of this transformation matrix. The deviation of this matrix from the identity matrix will be the basis of evaluating the mismatch between the full free-flying model and a simplified model in an initial study in simulation. Of course, feedback control can quickly compensate for small errors: to establish a reference figure, robot mass and inertia parameters are not usually known to an accuracy of greater than 5% (particularly inertia). Endpoint acceleration errors of this order of magnitude result in position errors small enough to be difficult to observe. If the control system is asked to respond within its bandwidth, the error can be expected to be $\frac{1}{K_p} \times \mathbf{a}_{error}$, where K_p is the position error control gain.

The errors in nonlinear terms f_{NL} will be zero if the base's *angular velocity* is accounted for. All of the nonlinear terms are functions of only the state (joint angles, angular rates), and are independent of the system accelerations $\dot{\mathbf{u}}_{1..n}$. In section 7.1.3 it was shown that inverse dynamics is a computationally inexpensive process compared to that of solving the Jacobian matrix equation. Thus, including the free-flying base's angular velocity here ensures accurate nonlinear feed-forward terms at very little increase in computational cost.

7.3.1 Variations in Base Mass and Payload Mass Parameters

Here the effects of neglecting base accelerations on a single-armed free-flying robot having the mass distribution of the experimental robot will be investigated using numeric simulation equations provided by **RD**. The CT control system is asked to deliver endpoint accelerations along the inertial $\hat{\mathbf{x}}$ and $\hat{\mathbf{y}}$ directions¹⁰ over two regimes: a range of robot base mass parameters, and a range of payload masses. Graphs are then plotted that show the values of the transformation matrix over a range of parameters.

¹⁰ Accelerations $\mathbf{a} = [1,0]$ and $[0,1]$ in Cartesian coordinates.

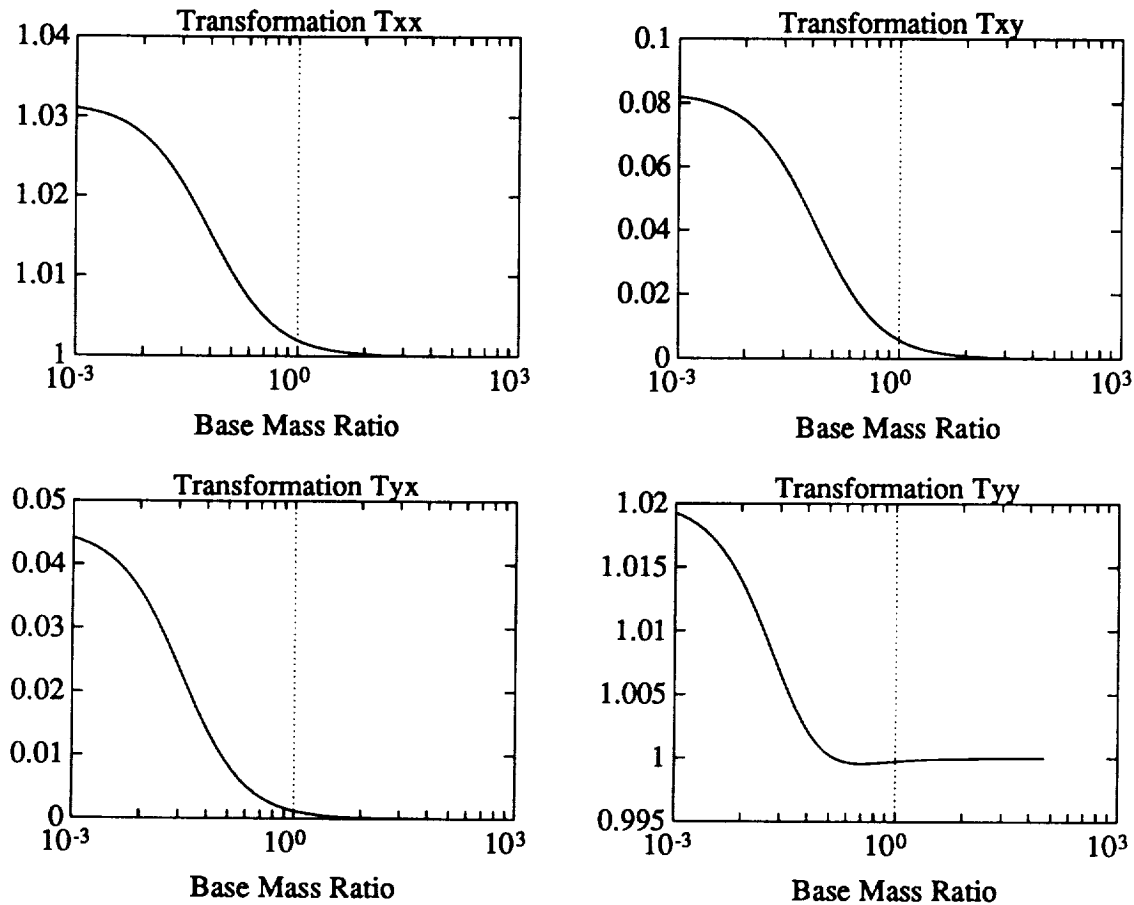


Figure 7.2:

**Endpoint Acceleration Deviations
from the Norm when Neglecting Base Accelerations, as a
function of Base Mass Parameters**

This plot depicts the deviation from an ideally unity transformation matrix between commanded and actual endpoint accelerations. When the mass and inertia are very large, the system behaves as if it is fixed-base. As the mass and inertia of the base decrease, the fixed-base approximation holds less well. Finally, as the mass and inertia of the base become very small, the larger modeling errors result in a small amount of cross-coupling between endpoint x and y accelerations. However, the transformation never deviates substantially from unity, because the lower arm has significant mass away from the elbow. The case of nominal mass distribution, that of the experimental robot, is indicated by the vertical lines.

Base Mass

The effect of simplifying the control system on the transformation matrix over a wide range of base mass (and inertia) values is shown in figure 7.2. Base mass parameters (mass and inertia) are varied with respect to those of the experimental nominal robot – 50 kg and 2.5 kg-m^2 – by a factor that varies between 10^{-3} and 10^3 . Actual manipulator endpoint acceleration response T_{xx} (in the x direction) deviates no more than 3% from the ideal for a very wide range of base masses. Similarly, actual endpoint acceleration response T_{yy} (in the y direction) deviates no more than 2%. Cross-coupling between x and y endpoint accelerations, T_{xy} and T_{yx} , is low - less than 8% in extremes, less than 1% for most of the range. These deviations are very small: it is not likely that they would even be noticed once a feedback controller is engaged. Friction effects and system mass parameter uncertainty are however undoubtedly greater than this.

Payload Mass

The effect of simplifying the control system on the transformation matrix over a wide range of payload mass values is shown in figure 7.3. In this case, payload mass introduced at the end of the manipulator arm is varied between 10^{-2} and 10^2 kg . Actual manipulator endpoint acceleration response T_{xx} (in the x direction) deviates no more than 0.5% for a very wide range of payload masses. Similarly, actual endpoint acceleration response T_{yy} (in the y direction) deviates no more than 0.1%. Cross-coupling between x and y endpoint accelerations, T_{xy} and T_{yx} , is very low - typically less than 0.6%. Interestingly enough, as the mass of the payload increases, the errors due to neglecting base accelerations *decrease*. This is because as the payload mass increases, the manipulator dynamics become less significant. The variations in the transformation matrix T are clearly negligible in this range.

These endpoint acceleration errors will have virtually no effect on the ultimate response of the endpoint control system. Experiments that compare the performance of a full-model CT endpoint controller versus a simplified controller should not reveal any noticeable change in response. Within the bandwidth of a feedback control system the position error can be expected to be $\frac{1}{K_p} \times \mathbf{a}_{\text{error}}$, where K_p is the position error control gain.

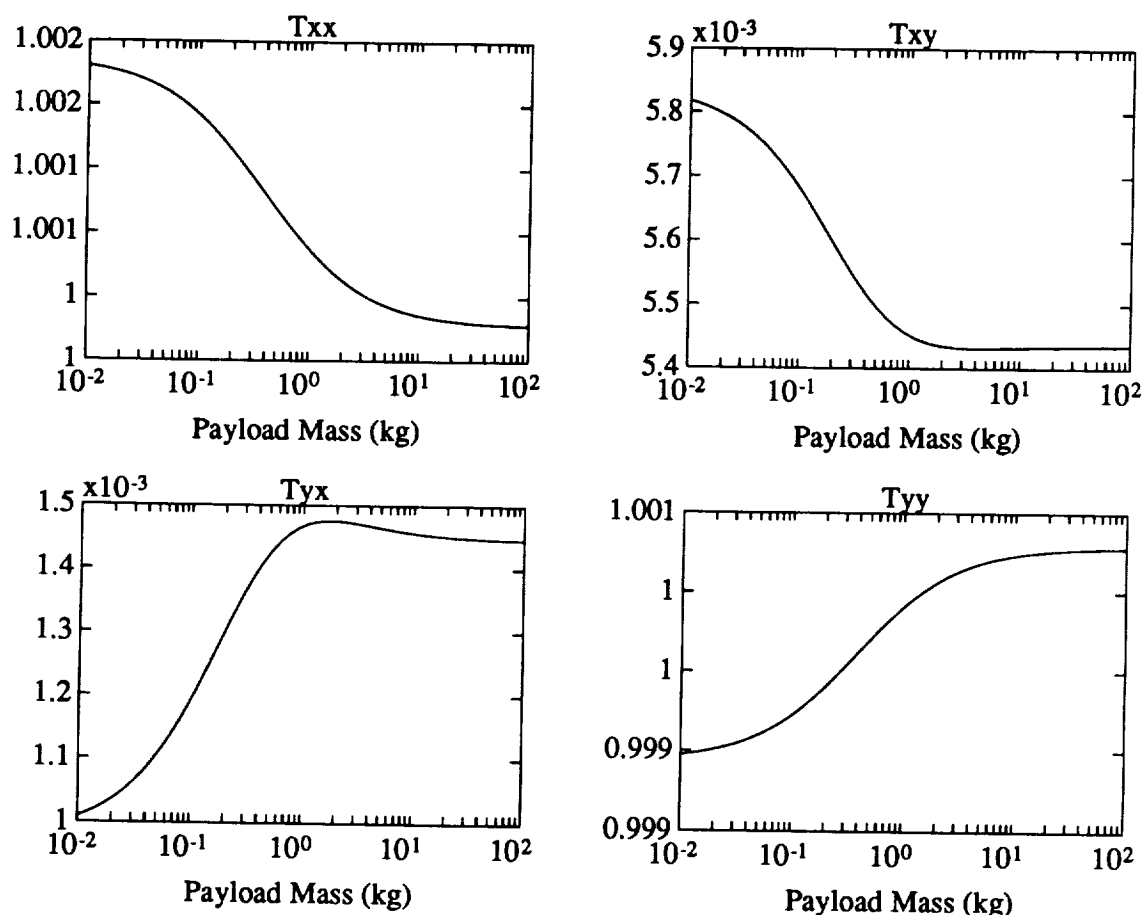


Figure 7.3:

Endpoint Acceleration Deviations from the Norm when Neglecting Base Accelerations, as a function of Payload Mass

This plot depicts the variation of an ideally unity transformation matrix between commanded and actual endpoint accelerations. When the payload mass is very large, the system behaves as if fixed-base. As the mass of the payload decreases, the fixed-base approximation holds less well. The transformation never deviates substantially from unity. The nominal mass distribution, that of the experimental robot, is indicated by the dashed vertical lines.

7.4 Experimental Results

Neglecting base accelerations of a free-flying robot may result in no noticeable decrease in performance of the controller, depending on the mass distribution in the dynamic system. The mass distribution of the experimental robot is as described in section 6.1: the base is massive (50 kg) compared to the manipulator arms (2.2 kg). In this section, two experiments performed with this robot demonstrate that, for its mass characteristics, both two-arm manipulator endpoint control and cooperative-arm object manipulation from a free-flying robot suffer no degradation in performance from using the simplified CT control system. The errors introduced by neglecting the base accelerations are very small compared to other modeling errors such as joint friction and spring forces, and uncertainty in the mass distribution (on the order of 5%).

7.4.1 Manipulator Endpoint Control

In this experiment, the positions of the two manipulator endpoints are position controlled. One is controlled to a fixed location, the other to follow a circular trajectory (just as described in section 6.1).

The expected acceleration errors, when read off of the graph in figure 7.2, for the base mass and inertia parameters of the robot (unity), are clearly less than a percentage point off in all the terms. The transformation matrix expressing actual endpoint accelerations in terms of desired accelerations is approximately

$$\mathbf{T} = \begin{bmatrix} 1.004 & 0.008 \\ 0.003 & 1.000 \end{bmatrix} \quad (7.12)$$

It should be difficult to observe any difference in endpoint error when the simplified endpoint controller is used. This is evident in figure 7.4, where a simulation of the experimental robot shows that modeling errors due to the simplified controller result in endpoint error on the order of 100 microns. This simulation employed a feedback controller using the same gains as those in previous endpoint experiments (section 6.1). The magnitude of this controller mismodeling is equivalent to an error in the mass and inertia estimates of the robot of the same magnitude: $\frac{1}{2}\%$.

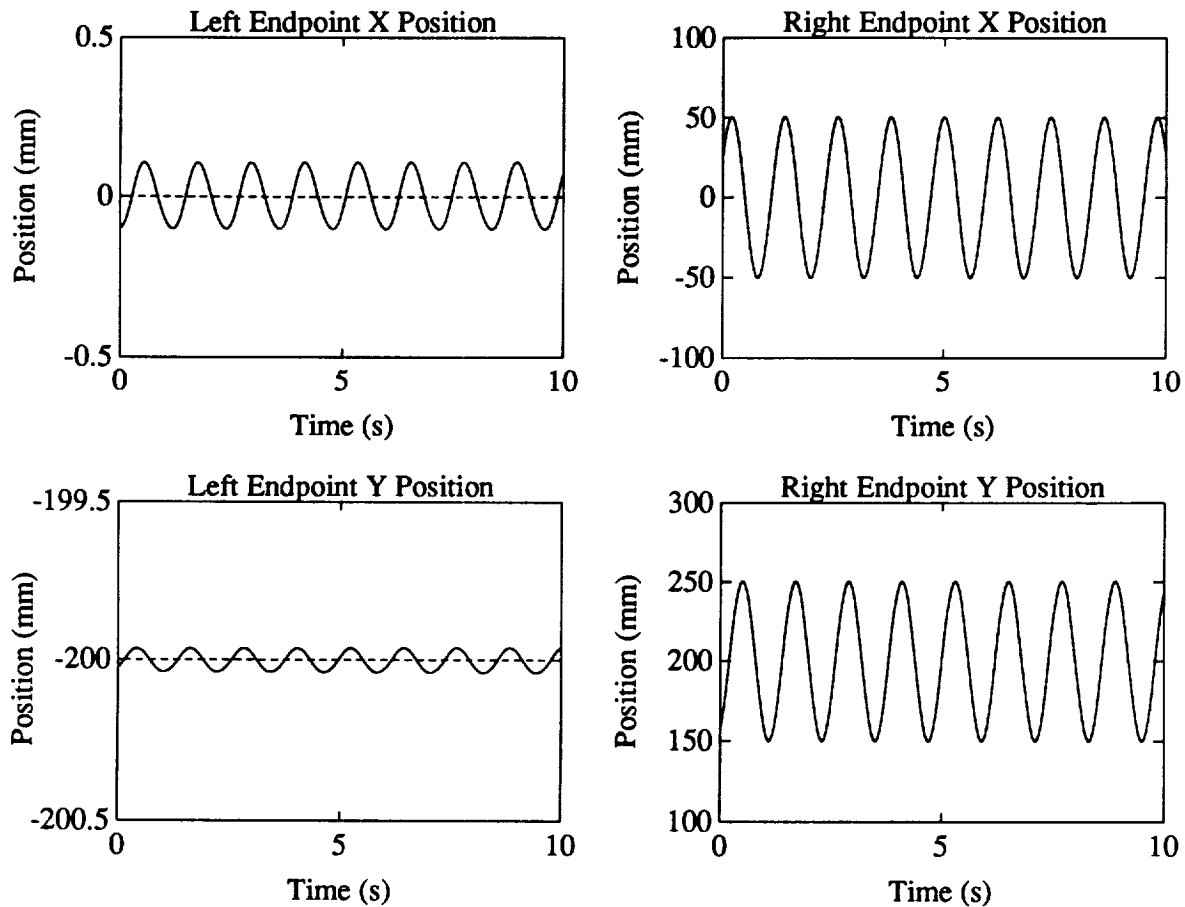


Figure 7.4: Expected Endpoint Controller Performance when Neglecting Free-Flying Robot Base Accelerations

This particular simulation predicts that neglecting robot base accelerations in the endpoint feedback control system of the experimental robot should result in very small position errors – on the order of 50-100 microns for the left arm endpoint. The errors due to neglecting base accelerations (endpoint acceleration errors of less than 1 are less than those due to uncertainties in mass distribution (on the order of 5%).

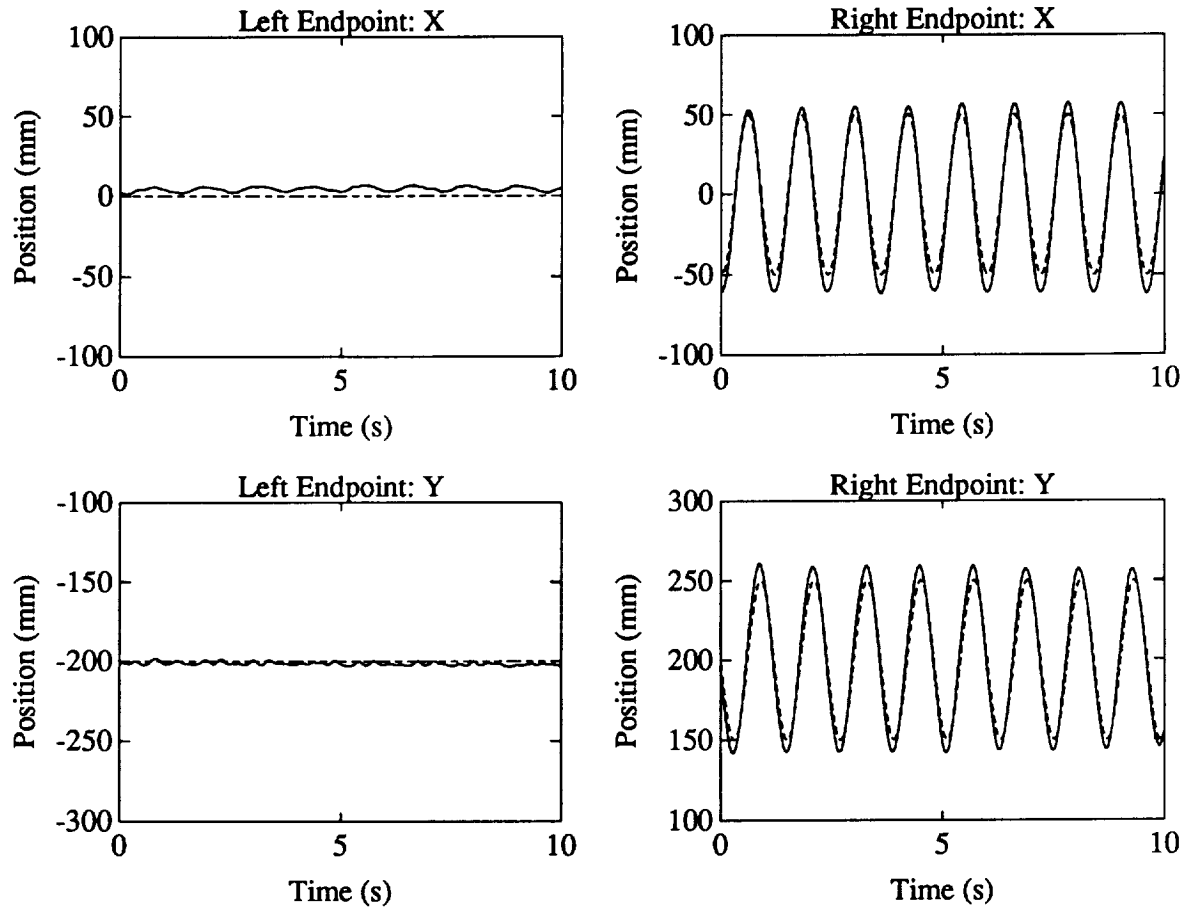


Figure 7.5: **Two-Arm Tracking Controller Data where Base Accelerations are Neglected in the CT Controller**

This experimental data shows a free-flying robot that neglects robot base accelerations in its CT control system. The CT controller does, however, compensate for the nonlinear effects of base angular velocity.

Figure 7.5 shows experimental data similar to that presented in section 6.1, page 94 – it uses an identical dynamic model in the kinematics and inverse dynamics. In this case, however, the CT controller neglects robot base accelerations. The tracking performance of this controller, like the full-model CT controller, is quite satisfactory. Endpoint errors for full-model CT control and simplified-model CT control are shown in figure 7.6. The errors visible are on the order of 5 mm peak-to-peak in both cases. They are primarily due to spring forces in the joints due to wiring and tubing. There was no noticeable difference in

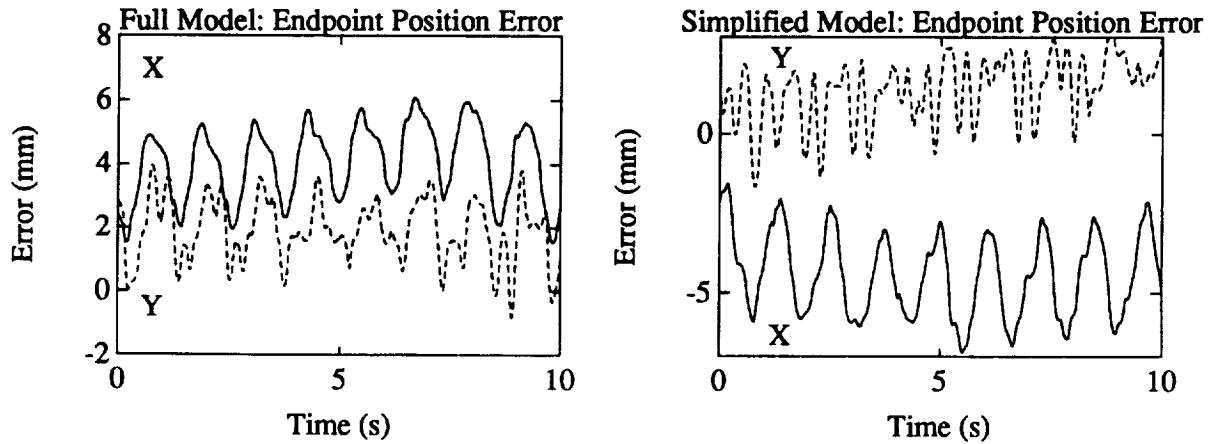


Figure 7.6: Differences in Endpoint Position Error between Including and Neglecting Free-Flying Robot Base Accelerations

This data shows that there is little noticeable change in error if a CT endpoint controller neglects robot base accelerations. The position error is of the left arm endpoint, presented in figures 6.4 and 7.5. X error is the solid line, Y error is the dotted line. Errors due to difficult-to-model wiring spring and friction forces exceed those due to not modeling base accelerations for this robot.

the performance of the two endpoint controllers.

7.4.2 Object Control

In this experiment, the positions of an object grasped by both manipulator arms will be controlled. The object's orientation is controlled to a constant angle, and its centerpoint is controlled to follow a circular trajectory (just as described in section 6.2).

The expected object acceleration errors, when read off of the graph in figure 7.3, for a payload mass of 6.9 kg are clearly less than a percentage point off in all the terms, in fact, most of them are on the order of 0.1% off. This graph (figure 7.3) also shows that as the payload mass increases, the modeling error *decreases*. The modeling error, already negligible in the previous example, will be even smaller here. The transformation matrix

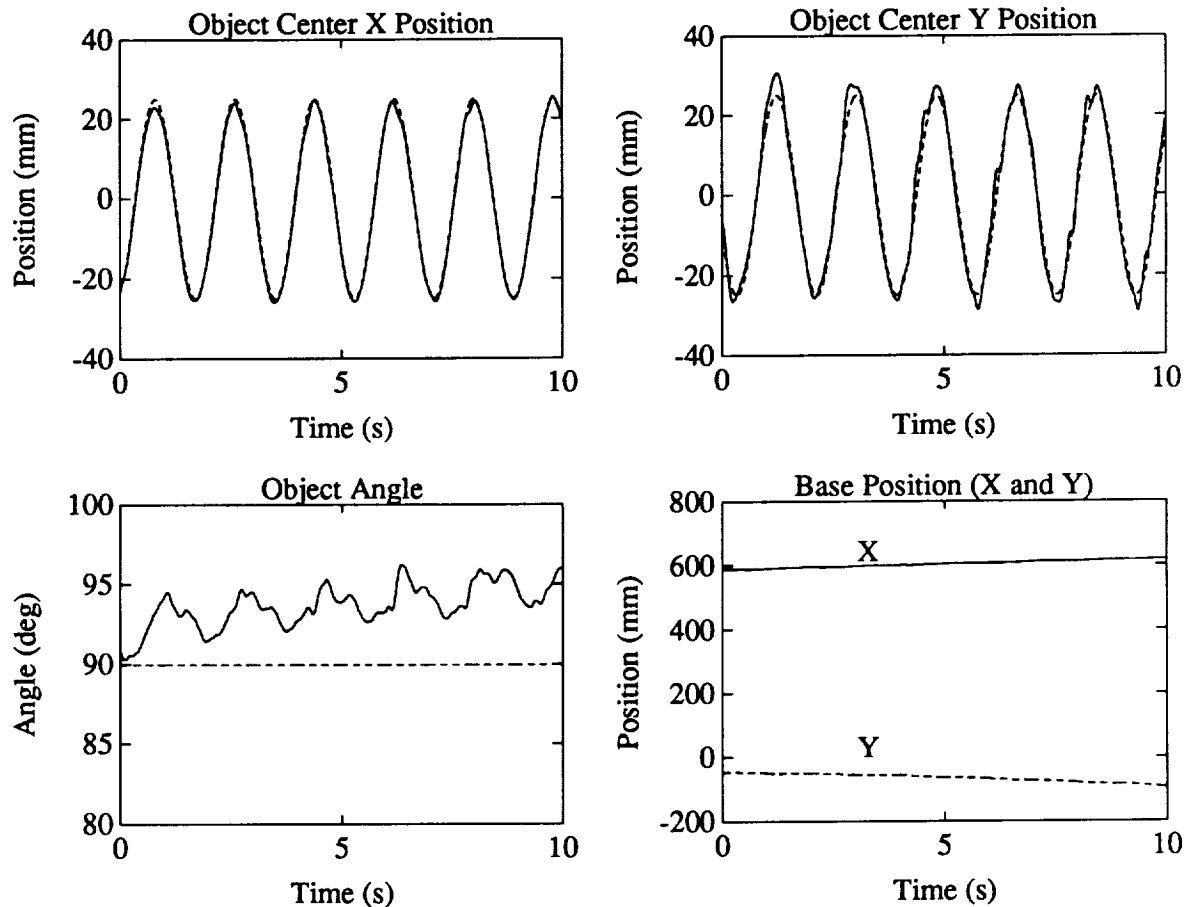


Figure 7.7: **Object Controller Data where Base Accelerations are Neglected in the CT Controller**

This experimental data shows the performance of a CT control system that neglects robot base accelerations on a free-flying robot. The CT controller does, however, compensate for the nonlinear effects of base angular velocity. Residual error is not substantially different from that of a CT controller that includes base accelerations: a comparison follows in figure 7.8. Since the object's center position is inferred from the grasp points, mechanical play causes slight jumps in estimated position.

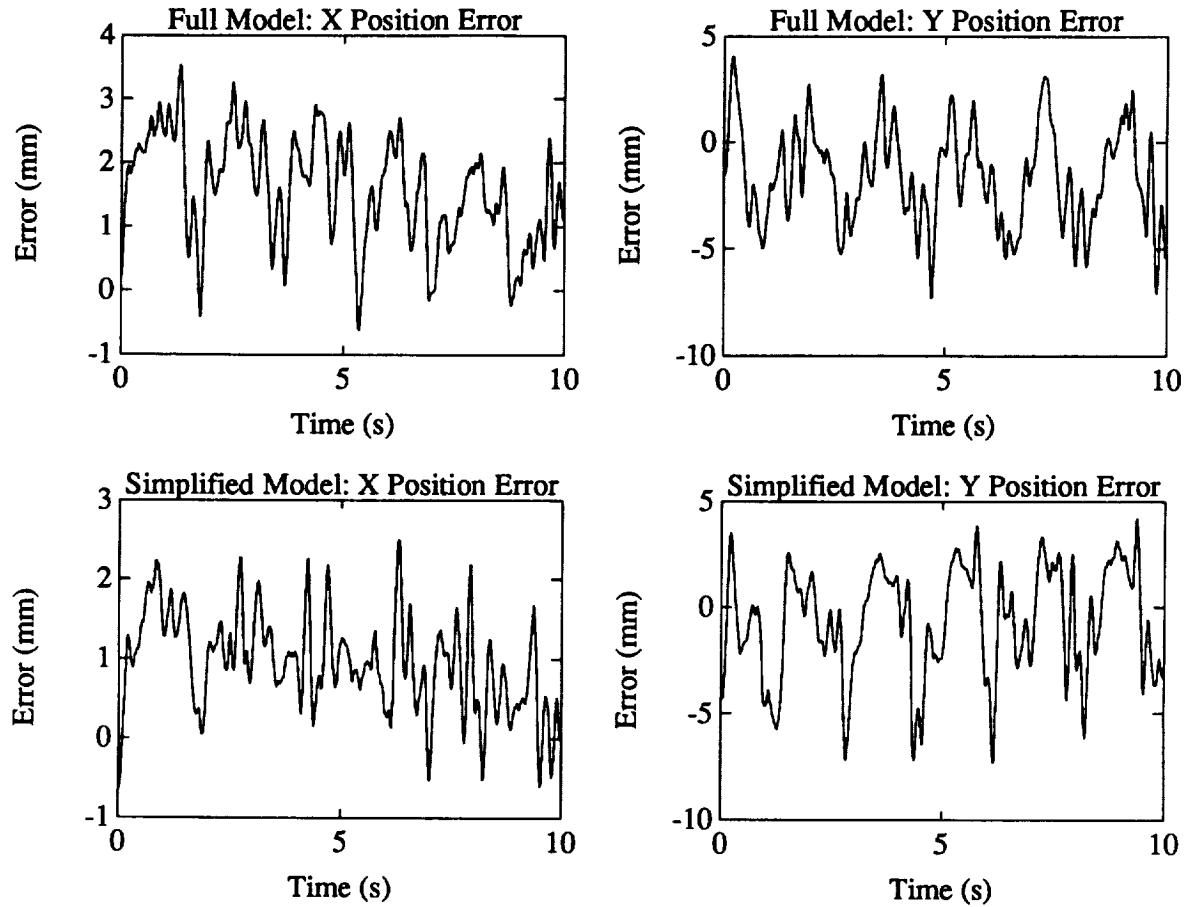


Figure 7.8: Differences in Object Positioning Error between Including and Neglecting Free-Flying Robot Base Accelerations

This data shows that there is little noticeable change in error if a CT object controller neglects robot base accelerations. Errors are mainly due to difficult-to-model wiring spring and friction forces.

expressing actual object accelerations in terms of desired accelerations is approximately

$$\mathbf{T} = \begin{bmatrix} 1.0005 & 0.0054 \\ 0.0015 & 1.0007 \end{bmatrix} \quad (7.13)$$

when neglecting the base accelerations. As was the case in the previous example, modeling errors this small result in negligible endpoint errors when under feedback control.

Figure 7.7 shows experimental data similar to that presented in section 6.2, page 104. It uses an identical dynamic model in the kinematics and inverse dynamics; but in the case, of

Figure 7.7 the CT controller neglects robot base accelerations. The tracking performance of this controller, like the full-model CT controller, is quite satisfactory, although object orientation could be controlled more tightly with a better angular-rate sensor (as with the full-model CT controller).

Comparisons between endpoint errors for full-model CT control and simplified-model CT control are shown in figure 7.8. The errors visible are on the order of 4 mm peak-to-peak for x and 10 mm peak-to-peak for y in both cases. This difference however is an artifact of the vision sensor, not the controller: the vision sensor has different resolutions in the x and y directions. Cyclic errors are primarily due to spring forces in the joints due to wiring and tubing. There was no noticeable difference in the performance of the two CT controllers.

7.5 Extremes in Mass Distribution

In order to truly see the limitations of simplified CT endpoint controllers on free-flying robots it is necessary to have a rather unusual mass distribution: one where the robot base mass is equal to or less than the manipulator's mass. In particular, the base mass and inertia of the base must be low, and the mass distribution of the link nearest the base must have little mass near the base. This unfavorable mass distribution results in large base accelerations occurring in response to manipulator endpoint activity. The effects of neglecting base accelerations over a range of base mass values for this fictitious class of robot will be examined in simulation. This robot has nominal mass characteristics listed in table 7.1.

Figure 7.9 shows the transformation matrix values (as shown initially in figure 7.2) over a range of base masses. This ideally unity transformation matrix does undergo substantial deviations from unity: at low base mass values there will be significant errors in real versus commanded endpoint accelerations if the base accelerations are neglected in the CT controller.

In addition, a measure of the condition number of the *full free-flying* augmented Jacobian matrix is plotted in figure 7.10. The condition number of the Jacobian is bad for both very small (200 g) and very large (40 kg) values of base mass, compared with the

manipulator of 2 kg. When the condition number increases, the ratio of the largest singular value of the matrix is substantially greater than the smallest. In practical terms, the base accelerations are significantly larger than the manipulator's at small base mass values, and the base accelerations are significantly smaller than the manipulator's at large base mass values.

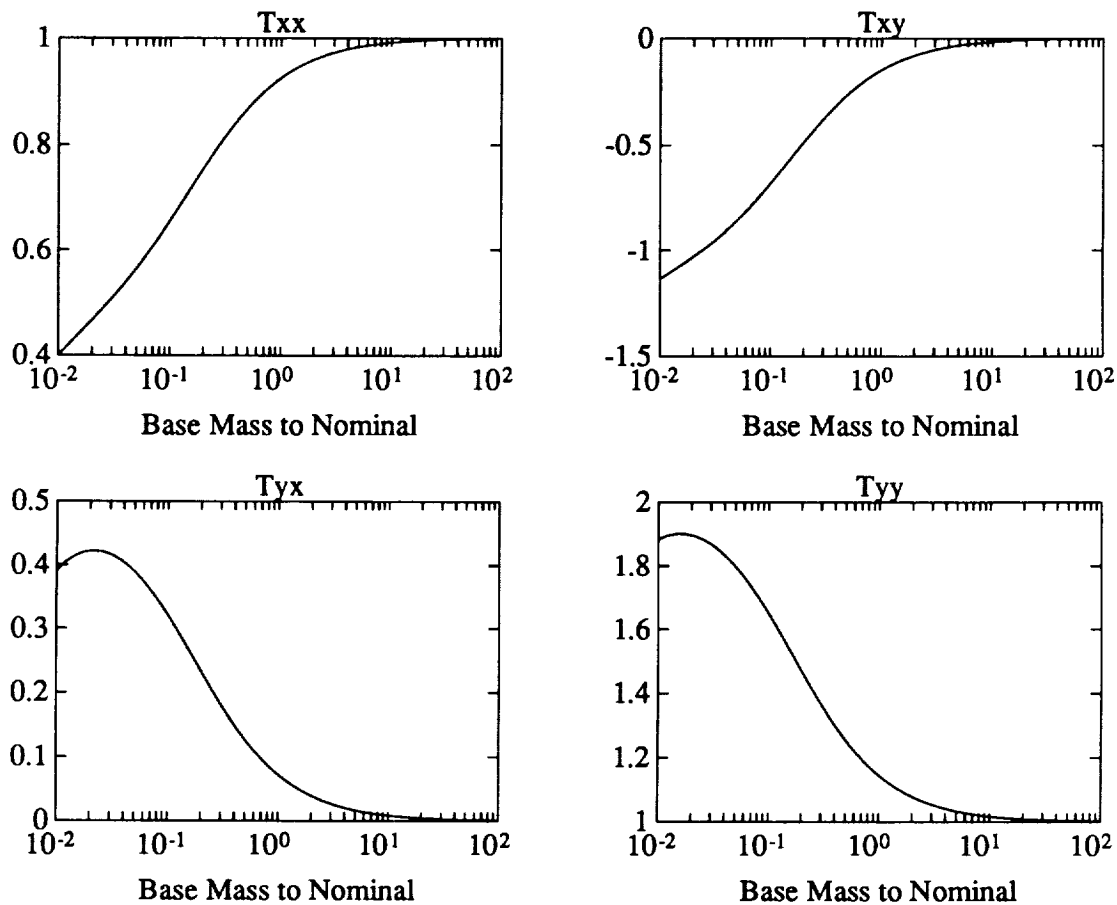


Figure 7.9: **Endpoint accelerations in Response to Commanded Accelerations as a Function of Base Mass.**

The inboard arm link has little mass at the shoulder, relying on the robot bases mass and inertia to 'anchor' the manipulator. As the base effectively disappears at low masses, the transformation matrix deviates substantially from unity.

It is evident from figures 7.9 and 7.10 that neglecting base accelerations divides into

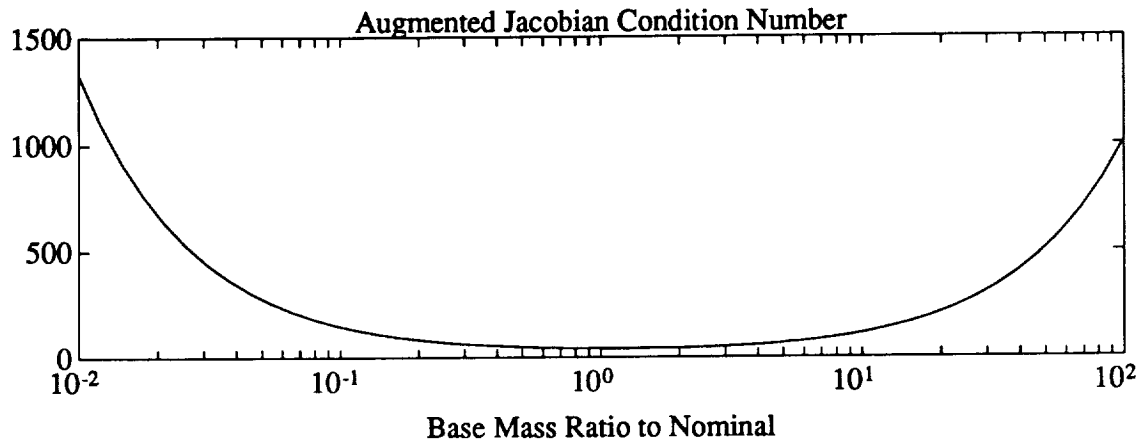


Figure 7.10: **Condition Number of Augmented Jacobian as a Function of Robot Base Mass.**

Notice that the condition number of the Jacobian is bad for both very low and very high base mass and inertia. With low base mass and inertia the base accelerations are very high, and with large base mass and inertia the base accelerations are very low.

three regimes for this class of robot: the first area is where the base mass and inertia are very small compared to the manipulator; the middle ground is where the base mass is comparable to that of the manipulator; and the rest is where the base mass is very large in comparison with the manipulator. The performance of controllers that neglect base accelerations operating in each of these regimes are discussed.

7.5.1 Very Small Base Mass and Inertia

At the extreme of zero base mass and moment of inertia, this free-flying manipulator degenerates into a single body (it's forearm) onto which torque can be exerted. The endpoint motion would not be controllable in all its degrees of motion. With small but non-zero base mass and inertia values, the base accelerations will be large in response to manipulator motions. Neglecting these accelerations will result in noticeable endpoint controller

performance degradation. A simulation run of a robot with base mass $\frac{1}{10}$ that of the manipulators (and correspondingly $\frac{1}{10}$ the moment of inertia) is shown in figure 7.11. Small endpoint motions do result in large base motions. The CT controller includes a feedback section that has insufficient bandwidth to compensate for the motions commanded: the feedforward characteristics of CT control are being relied on.

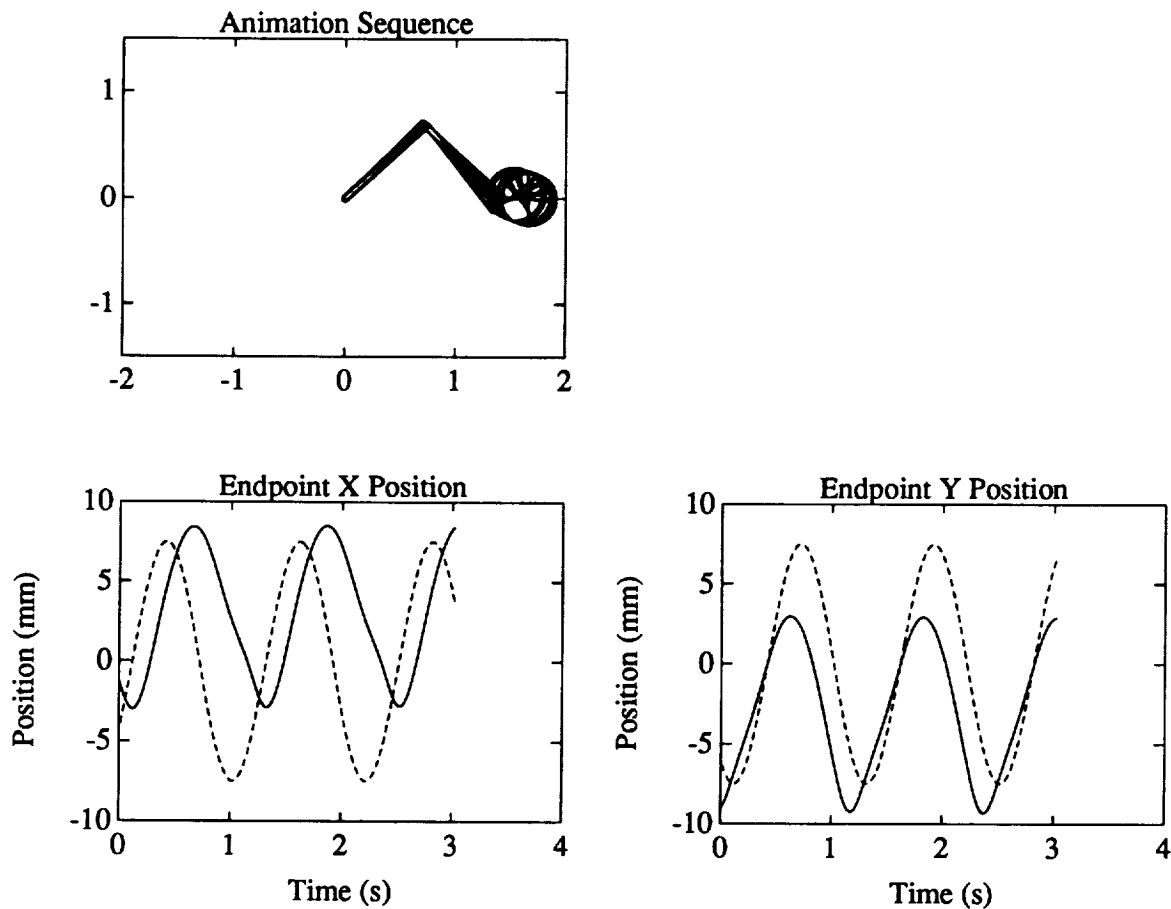


Figure 7.11: **Robot with Light Base Mass Attempting Endpoint Motion**

When the robot base is very light ($\frac{1}{10}$ the mass of the manipulator), the ability of the manipulator to move its endpoint is very limited. Neglecting base accelerations, as done in this controller, results in significant endpoint error: it is necessary to include base accelerations in the controller for such a robot.

The poor performance of the simplified endpoint CT controller shows that it is necessary

to include the base accelerations in the model. The transformation matrix expressing actual endpoint accelerations in terms of desired accelerations is approximately

$$\mathbf{T} = \begin{bmatrix} 0.6 & -0.8 \\ 0.4 & 1.8 \end{bmatrix} \quad (7.14)$$

The transformation matrix deviates substantially from unity, up to 80%. Acceleration errors are well beyond the 5% acceptable as modeling error due to mass distribution uncertainties.

7.5.2 Medium Base Mass and Inertia

When the base mass and inertia are larger than the manipulator but not significantly so, it may or may not be necessary to account for base accelerations in the CT controller. Figure 7.12 shows simulation of a robot with mass (2 kg) equivalent to the manipulator (2 kg) performing endpoint motions. The base moves a fair amount in response to motions of the manipulator.

The errors in endpoint tracking are small but noticeable. The transformation matrix expressing actual endpoint accelerations in terms of desired accelerations is approximately

$$\mathbf{T} = \begin{bmatrix} 0.85 & -0.25 \\ 0.08 & 1.15 \end{bmatrix} \quad (7.15)$$

The transformation matrix deviates substantially from unity - about 15-25%. Acceleration errors are beyond the 5% acceptable as modeling error due to mass distribution uncertainties, but may be acceptable if performance requirements are met.

7.5.3 Large Base Mass and Inertia

When the robot base mass and inertia are large compared to the manipulator(s), as has been discussed in an earlier section and shown in two experiments, the base accelerations can be neglected. Neglecting the base accelerations in the controller is not only advantageous, it makes computation faster and numerically better conditioned (see figure 7.10 for Jacobian condition number).

In summary, the results presented in sections 7.4.1 and 7.4.2, including the endpoint controller performance prediction in figure 7.4, illustrate that for a practical space-based

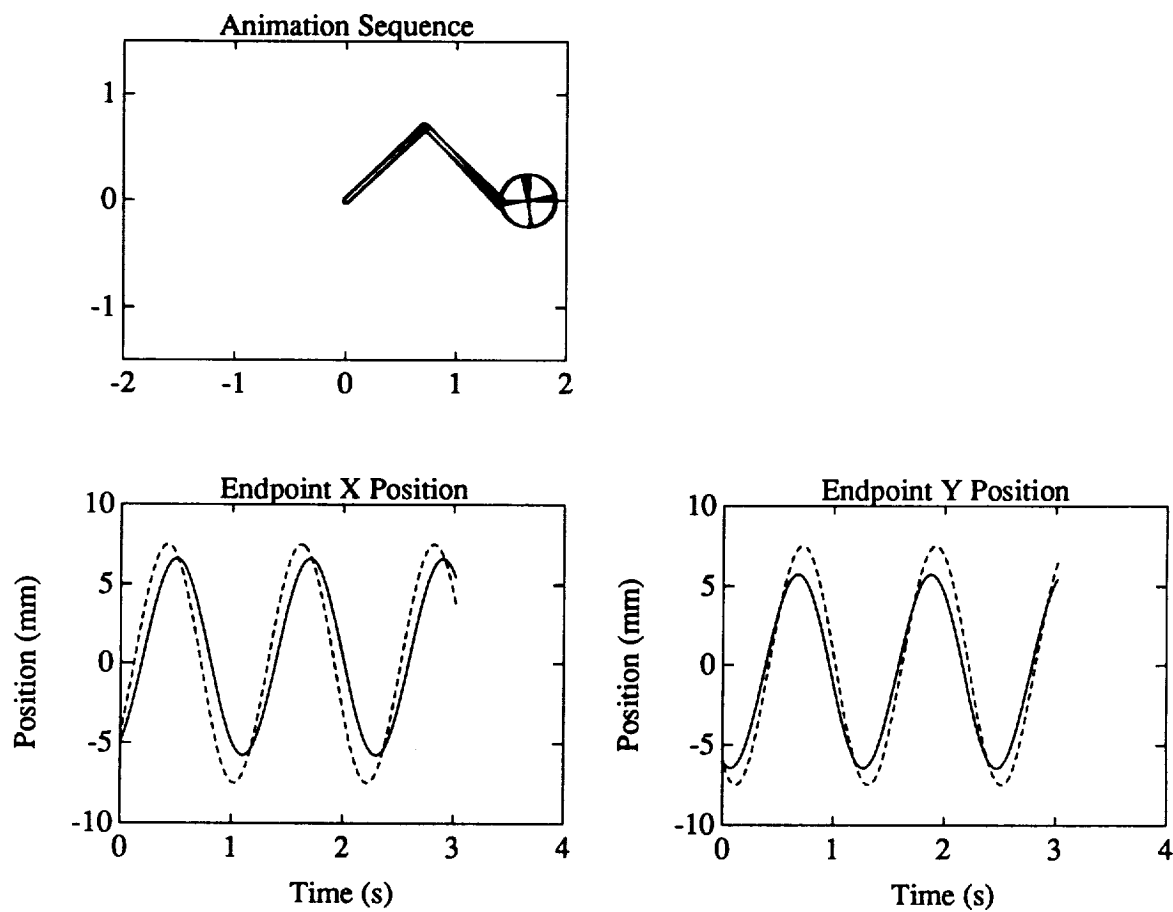


Figure 7.12: **Robot with Medium Base Mass Attempting Endpoint Motion**

When the manipulator is as massive as the robot base, the ability of the manipulator to move its endpoint is limited. Neglecting base accelerations, as done in this controller, results in a noticeable endpoint error: it may or may not be necessary to include base accelerations in the controller for such a robot.

robot – one containing propulsion, power, communication and control systems – it is not necessary to include base accelerations in the CT controller.

7.5.4 Manipulating Massive Payloads from Free-flying Robots

In figure 7.3, the plot of the transformation matrix between actual and real endpoint accelerations when neglecting base accelerations remains almost identically unity over a very wide range of payload masses. This shows that the *acceleration* of the payload can be controlled regardless of its mass. This does not imply, however, that the payload can be *moved* very far. Action-reaction can make the robot move a lot compared to the payload, and if no external forces or moments are applied to the robot body, the manipulators may exceed their joint range of motion and/or approach singular configurations.

A terrestrial example of such a system is a tugboat pulling on a large ship with a high-power winch. The ship can be made to accelerate by applying power to the winch. However, if the tug's engines (external forces) are not engaged, the winch will cease to be effective once the tug is pulled into the ship: external forces applied to the tug are necessary to maintain it at a reasonable operating distance.

Another example is a robot holding a comparably sized payload at "arm's length". While it is possible for the robot to push or pull on the object, it is not able to move it sideways very far without some kind of external moment to prevent the base of the robot from rotating out of range of the manipulator arms.

7.6 Summary

In this chapter, the consequences of neglecting the base accelerations in the CT controller of a free-flying robot were investigated. Base accelerations and base angular velocity are what differentiate fixed-base CT controllers from full-model free-flying robot CT controllers.

A space robot designer would be prudent to include base angular velocity compensation in the CT model, since it compensates for the substantial nonlinear inertial forces introduced by base angular velocity – and costs little to include.

The effects of neglecting robot base accelerations over a range of robot/payload mass-distributions were also examined. In only extreme cases – where the robot base and upper

arm had small mass and inertia compared to the forearm – did base accelerations play a significant role in assuring accurate control. The mismodeling due to neglecting base accelerations was frequently found to be significantly less than that due to uncertainty in the mass distribution (mass and inertia) in the robot (typically on the order of 5%).

The automated CT control computer program (**RD**) developed in chapter 4 was used to show that base accelerations played a minimal role in these and other control systems. This program can also be used to further investigate other manipulator configurations and robot/payload mass distributions to answer questions about the performance of specific systems not covered here.

Two experimental demonstrations of a free-flying robot moving its manipulators and moving a grasped object serve as examples to show that including base-accelerations in the manipulator CT controller may not be necessary. The endpoint controllers showed no noticeable change in performance when neglecting robot base accelerations. The savings in computations offered by neglecting base accelerations, combined with the negligible performance degradation under certain conditions make it a practical and beneficial technique for assuring high-performance endpoint control at reasonable computational cost.

Base Mass	2.0 kg
Base Inertia about c.m.	0.1 kg-m ²
Vector to c.m.	0.0, 0.0 m
Vector to Shoulder	0.25, 0.0 m
Lower Arm Mass	1.0 kg
Lower Arm Inertia about c.m.	0.01 kg-m ²
Vector to c.m.	0.5, 0.0 m
Vector to Elbow	1.0, 0.0 m
Upper Arm Mass	1.0 kg
Upper Arm Inertia about c.m.	0.03 kg-m ²
Vector to c.m.	0.0, 0.0 m
Vector to Endpoint	1.0, 0.0 m
Payload Mass	1.0 kg
Payload Inertia about c.m.	0.01 kg-m ²
Vector to c.m.	0.0, 0.0 m

Table 7.1: Nominal Mass Distribution in a Fictitious Robot

Vectors are expressed along local x and y axes: the x axis points toward the next joint. The configuration studied places the shoulder angle nominally at -45 deg and the elbow at 90 deg.

Chapter 8

Conclusions

8.1 Summary

A new method for integrating momentum control into computed-torque controllers has been introduced: it places momentum into an augmented Jacobian matrix (equation 3.1). This both simplifies the process of constructing a square Jacobian matrix for computed torque, and reduces the amount of computation needed to solve the problem when compared to the Generalized Jacobian technique of Umetani and Yoshida.

This idea has also been applied to dynamic constraints, such as closed-chain constraints, by augmenting the Jacobian with a very simple constraint relation. Similar benefits result: simpler construction of the Jacobian matrix, and fewer required computations. A major benefit of this approach, when used with constrained dynamic systems, is that it is not necessary to formulate constrained equations of motion for the computed-torque controller.

By using the Jacobian augmentation approach, the computed-torque formulation extends naturally to free-flying and closed-chain robot configurations.

In addition, a new method for formulating equations of motion for *simulation* of constrained dynamic systems was presented. These simulation equations (equations 2.45) ensure that constrained systems converge on a consistent state despite small initial state errors, via numerical relaxation. The buildup of numerical error (a violation of constraints) is also prevented.

A recursive algorithm for calculating kinematic quantities for rigid multibody robots

was presented. Elements of the Jacobian matrix and the inverse dynamics were expressed in terms of partial velocities. This algorithm was implemented as a computer program (RD). It automatically recursively calculates the terms in and solves the Jacobian equation and evaluates the motor torques in a Newton-Euler inverse dynamics routine. Using this program, it is possible to implement a controller simply by specifying the dynamic system, the desired quantities in the Jacobian equation, and any unmeasured signals that need to be calculated. No hand derivations are required. This powerful tool may be used both as a real-time controller and also as a dynamics and control simulation tool.

A laboratory robot was designed and constructed¹ to act as a testbed for evaluating control systems. This robot, like NASA's proposed Orbital Maneuvering Vehicle, has a large base body flying freely², housing power and propulsion systems, and having two lightweight cooperating robot arms for manipulation.

The RD program was used to implement independent arm endpoint control and also cooperative-arm object manipulation. RD was also used to simulate the behavior of these multibody systems. Full free-flying multibody dynamic models were used in experimental demonstrations of endpoint control of a free-flying robot with two arms. Arm endpoint control and cooperative-arm object control were demonstrated.

An experimental examination of the effects of neglecting free-flying robot base *accelerations*³ in an endpoint feedback controller was made. In the physical systems examined, this simplification offers little degradation in system performance in exchange for a large savings in computation (robot base *angular velocity* was needed to compensate for the nonlinear effects of base rotation, and cost little in computation). Further simulations were used to explore over what range of situations this would be valid: i.e. where the errors due to mismodeling resulted in errors of accelerations of less than 5%. These errors are further reduced if within the bandwidth of the feedback control system. The interesting result is that this simplification is valid over a very large regime. A tool such as RD is both very useful and necessary for investigating specific systems: it is difficult to generalize results in nonlinear systems.

¹with Marc Ullman

²Flying freely in two dimensions.

³Base accelerations were neglected, not angular motions. Angular velocity needs to be included in order to compensate for centripetal forces.

8.2 Recommendations for Further Research

While several aspects of endpoint control from a free-flying robot were examined in this thesis, they have all dealt with the robot in pure free flight: no external momentum control devices were used. Even although it was shown that manipulator control and object manipulation are possible from a free-flying robot with no momentum control, it is very evident from both simulation and experiment that the workspace of the manipulators can easily be exceeded, due either to initial motion, or to activities of the manipulators. Providing simultaneous control of the base motions via reaction wheels and/or thrusters would allow the robot to maintain its workspace.

The methods presented here for simplifying the formulation of computed-torque controllers reduce the amounts of computation over previously discussed methods; however, it is still computationally expensive $\mathcal{O}(n^3)$ to invert the Jacobian equation in order to solve the control problem. Inversion is a numerically expensive process. In newer dynamics simulation tools [27] and techniques [25], [26], it is possible to perform dynamic system simulation in order n computations. Extending this work to solve the related but different problem of computed-torque control would be of considerable benefit to the controls community, offering numerical solutions for large dynamic systems at less computational cost.

In order for a computed-torque controller to function predictably, it is important to have a good model of the payload, which is frequently quite massive compared to the manipulators. It is far better to underestimate the mass parameters and get a slow computed-torque controller than to overestimate and get effectively higher error-controller gain values that can cause the system to become underdamped and possibly unstable. An adaptive controller that allowed progressive estimation of the payload mass would be a valuable asset to a space robot.

Appendix A

Power and Energy Expressions

A.1 Kinetic Energy

In a dynamic system of ν bodies as under discussion, the kinetic energy can be expressed as:

$$\begin{aligned} K &= \frac{1}{2} \sum_{i=1}^{\nu} \left(m^i \mathbf{v}^{i*2} + \omega^i \cdot \mathbf{I}^{i/i*} \cdot \omega^i \right) \\ &= \frac{1}{2} \sum_{i=1}^{\nu} m^i \mathbf{v}^{i*} \cdot \mathbf{v}^{i*} + \frac{1}{2} \sum_{i=1}^{\nu} \omega^i \cdot \mathbf{I}^{i/i*} \cdot \omega^i \\ &= \frac{1}{2} \sum_{i=1}^{\nu} m^i \left(\sum_{r=1}^n \mathbf{v}_r^{i*} u_r \right) \cdot \left(\sum_{s=1}^n \mathbf{v}_s^{i*} u_s \right) \\ &\quad + \frac{1}{2} \sum_{i=1}^{\nu} \left(\sum_{r=1}^n \omega_r^i u_r \right) \cdot \mathbf{I}^{i/i*} \cdot \left(\sum_{s=1}^n \omega_s^i u_s \right) \\ &= \frac{1}{2} \sum_{i=1}^{\nu} \left(\sum_{r=1}^n \sum_{s=1}^n m^i \mathbf{v}_r^{i*} \cdot \mathbf{v}_s^{i*} u_r u_s + \sum_{r=1}^n \sum_{s=1}^n \omega_r^i \cdot \mathbf{I}^{i/i*} \cdot \omega_s^i u_r u_s \right) \\ &= \frac{1}{2} \sum_{r=1}^n \sum_{s=1}^n m_{rs} u_r u_s \\ &= \frac{1}{2} \mathbf{u}^T \mathbf{M} \mathbf{u} \end{aligned} \tag{A.1}$$

A.2 Power

The power input is due to work done by the actuators (arm torquers), and forces exerted on the manipulator endpoints during contact with external objects. The power input into

a system of ν bodies as under discussion can be expressed as follows:

$$\begin{aligned}
 \mathbf{P} &= \sum_{\text{Applied Forces}} \mathbf{F}^{\text{Applied}} \cdot \mathbf{v} + \sum_{\text{Applied Torques}} \mathbf{T}^{\text{Applied}} \cdot \boldsymbol{\omega} \\
 &\stackrel{2.1}{=} \sum_{\text{Applied Forces}} \mathbf{F}^{\text{Applied}} \cdot \sum_{r=1}^n \mathbf{v}_r u_r + \sum_{\text{Applied Torques}} \mathbf{T}^{\text{Applied}} \cdot \sum_{r=1}^n \boldsymbol{\omega}_r u_r \\
 &= \sum_{r=1}^n \left(\sum_{\text{Applied Forces}} \mathbf{F}^{\text{Applied}} \cdot \mathbf{v}_r u_r + \sum_{\text{Applied Torques}} \mathbf{T}^{\text{Applied}} \cdot \sum_{r=1}^n \boldsymbol{\omega}_r u_r \right) \\
 &\stackrel{[13]}{=} \sum_{r=1}^n F_r u_r \\
 &= \mathbf{F}^T \mathbf{u}
 \end{aligned} \tag{A.2}$$

This is an simple and intuitively pleasing result.

A.3 Nonlinear Terms

The nonlinear terms have a simple relationship with the mass matrix that can be shown by evaluating power input to the system:

$$\begin{aligned}
 \mathbf{P} &= \mathbf{F}^T \mathbf{u} \\
 &= \frac{1}{2} \mathbf{F}^T \mathbf{u} + \frac{1}{2} \mathbf{u}^T \mathbf{F} \\
 &= \frac{1}{2} (\mathbf{M} \dot{\mathbf{u}} + \mathbf{N} \mathbf{u})^T \mathbf{u} + \frac{1}{2} \mathbf{u}^T (\mathbf{M} \dot{\mathbf{u}} + \mathbf{N} \mathbf{u}) \\
 &= \frac{1}{2} \mathbf{u}^T \mathbf{M} \dot{\mathbf{u}} + \frac{1}{2} \dot{\mathbf{u}}^T \mathbf{M} \mathbf{u} + \frac{1}{2} \mathbf{u}^T \mathbf{N} \mathbf{u} + \frac{1}{2} \mathbf{u}^T \mathbf{N}^T \mathbf{u} \\
 &= \frac{1}{2} \mathbf{u}^T \mathbf{M} \dot{\mathbf{u}} + \frac{1}{2} \dot{\mathbf{u}}^T \mathbf{M} \mathbf{u} + \frac{1}{2} \mathbf{u}^T (\mathbf{N} + \mathbf{N}^T) \mathbf{u}
 \end{aligned} \tag{A.3}$$

but power is also

$$\begin{aligned}
 \mathbf{P} &= \frac{1}{2} \mathbf{u}^T \dot{\mathbf{M}} \mathbf{u} \\
 &= \frac{1}{2} \mathbf{u}^T \dot{\mathbf{M}} \mathbf{u} + \frac{1}{2} \dot{\mathbf{u}}^T \mathbf{M} \mathbf{u} + \frac{1}{2} \mathbf{u}^T \dot{\mathbf{M}} \mathbf{u}
 \end{aligned} \tag{A.4}$$

and examining the difference, it is clear that

$$\dot{\mathbf{M}}_{\text{A.3,A.4}} = \mathbf{N} + \mathbf{N}^T \tag{A.5}$$

This result shows that the derivative of the mass matrix is symmetric, even if the nonlinear terms are not – which tends to be the case. Furthermore, if the mass matrix is constant, then

$$0_{A.3} = N + N^T \quad (A.6)$$

and the nonlinear matrix must be skew symmetric.

2

Appendix B

Circuit Diagrams

This appendix contains circuit diagrams for various subsystems of the satellite robot model. The following circuits are documented here:

- The Power Control Board
- The Battery Charging Board
- The Safety Disconnect Board
- The Inertial/Global Position Sensor Interface Board

The experimental vehicle also contains interface boards that are common to other experimental hardware in the Stanford Aerospace Robotics Laboratory. These circuits are not documented here.

- The Motor Driver Board (delivers commanded current to motor)
- The RVDT Board (provides angle and rate estimate)
- The Force Sensor Board (provides force estimate)

B.1 Power Control Unit

The power control unit provides basic control functions for power on the vehicle. The master power switch (figure B.1) controls all power to the analog and computer circuits.

When off, the only thing that will operate are the battery chargers. When on, external power (figure B.2), if connected, will be engaged onto the main power bus. If external power is not applied, switches along the front panel allow batteries to be individually engaged onto the power bus (figure B.3).

LEDs indicate whether batteries are available, and whether they are engaged onto the power bus. It is possible for the on-board computer to determine whether or not batteries are engaged via TTL level output signals provided via opto-couplers. Provision also exists for the computer to control the batteries using TTL level signals.

The two tiny red LED's indicate when the power bus is asymmetrical: this can occur if batteries are discharged, battery fuses are blown, or external power is not correctly connected.⁴ Battery fuses are located on the battery board. The circuit board connector pinout is documented in figure B.4.

B.2 Battery Charging and Monitoring

This circuit provides battery charge capability on-board the robot. When external power is connected, the batteries will charge in a manner determined by the settings of the charge switch (figure B.5).

Batteries can be charged by external power while engaged onto the power bus. Signals available to the computer include battery voltage and current, both during charge and discharge.

Green LEDs on the front panel indicate the relative amount of current being drawn from the battery, their brightness is proportional to the discharge current: bright green is 10 A. An LED is provided for each of the two 12 V batteries making up a $\pm 12V$ power source.

Red LEDs on the front panel indicate the relative amount of current with which the battery is being charged: dim is about 150 mA, medium is about 700 mA, and bright is about 1.5 A.

There are two fuses per battery: a charge fuse to limit charge current to 2A, and a discharge fuse to limit discharge current to 15A. If the discharge fuses blow (typically one goes, the other doesn't), one of the bus fault LEDs on the PCU will activate. If the charge

B.2. Battery Charging and Monitoring

153

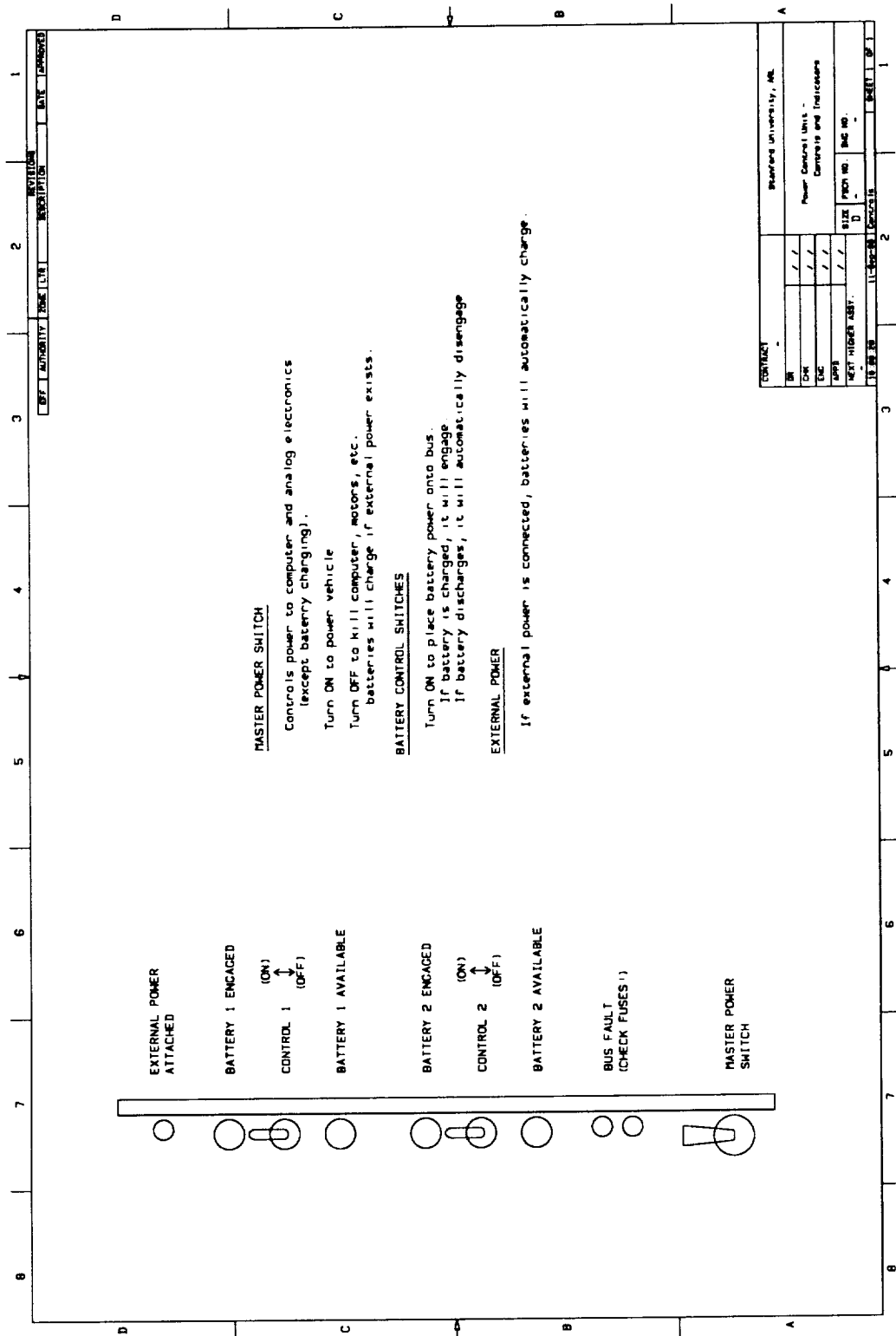
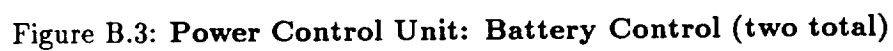


Figure B.1: Power Control Unit: Switches and LEDs

ORIGINAL PAGE IS
OF POOR QUALITY

155



ORIGINAL PAGE IS
OF POOR QUALITY



ORIGINAL PAGE IS
OF POOR QUALITY

fuses blow, the batteries won't charge. A problem with the current design is that the charge fuses will tend to blow if batteries are plugged in while external power is connected.

B.3 LEDs and Inertial Sensors

The robot has marker LEDs visible from above so that the vision system can track the two arm endpoints. The endpoint LED's are powered via resistors on a circuit board devoted to miscellaneous functions (figure B.6).

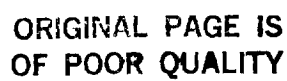
The angular rate sensor provides a measurement of the angular rate of the robot's base body. It requires some filtering before conversion to digital form in order to avoid aliasing, and also to remove the substantial amount of noise occurring above 10 Hz.

B.4 Safety Disconnect Board

The robot has a safety cutout circuit that disconnects the motors and thrusters from the computer: effectively preventing the robot from doing anything. It operates in one of two modes: one that indicates that the user trusts the computer, and the other mode where the user does not trust the computer. This mode is set by the one on/off switch on the board: up trusts the computer, down does not. There are two push button switches – the lower one activates the safety system, turning on the relays if the computer heartbeat is active, and the upper switch kills the system. An external kill switch can also be connected to the robot via a small plug located near the external power connector.

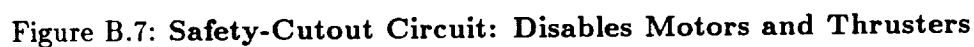
If the computer can be trusted, then the heartbeat signal coming from the computer (a square-wave signal at about 30 Hz) will activate all the relays in the system, making it live. It can be killed via user control (the kill switch), but will be dead only as long as the switch is held. The computer heartbeat can be disabled (in software) to achieve the same effect. This mode is typically used once things are working well.

If the computer (or controller) is not to be trusted, then the user must activate the safety system manually using the lower push button. The system can be killed by pushing the kill switch (or the external kill switch). It will not automatically reactivate. This mode is typically used when first testing out controllers.



The circuit diagram is shown in figure B.7.

161



ORIGINAL PAGE IS
OF POOR QUALITY

2

Appendix C

RD Computer Software

The interfaces to the **RD** software package are documented here:

- The C language interface specification
- The Matlab interface specification

The following components of the **RD** software package C source code are documented here:

- The Evaluation of Kinematics
- The Evaluation of Dynamical Equations of Motion

the following components are not included, but can be obtained from the author:

- The Input File Parser
- The Initialization of Data Structures
- The Construction of the Jacobian Equation
- The Matrix Inversion Solution
- The C Library Interface Implementation
- The Matlab Interface

C.1 C Language Interface Specification

This interface specification comes from the include file *rd.h*.

```

/*
 * @(#)rd.h    generated by: makeheader    Thu Aug 23 15:02:50 1990
 *
 * built from: robot2d.h
 * rd.c
 * print.c
 */

#ifndef rd_h
#define rd_h

typedef struct ROBOT_STRUCTURE ROBOT;

typedef struct RD_SIGNALSET_STRUCTURE
{
    int      n;           /* number of entries          */
    char     **names;     /* names of entries           */
    char     **units;     /* units of entries           */
    int      **isValid;   /* pointer to array of ptrs to logical */
    float     **values;   /* pointer to array of ptrs to entries (USER) */
} RD_SIGNALSET;

typedef struct RD_DATASET_STRUCTURE
{
    char      *name;       /* user supplied name         (USER) */
    char      *filename;   /* configuration filename      */
    ROBOT      *robotP;    /* internal robot data structure */

    int      p;           /* number of degrees of freedom */
    int      c;           /* number of constraints        */
    int      n;           /* number of generalized coords (p+c) */

    RD_SIGNALSET  genCoords; /* Generalized Coordinates          */
                                /* number of coords                 */
                                /* names of coords                  */
                                /* units of coords                  */
                                /* array of pointers to coords      (USER) */

    RD_SIGNALSET  genCoordsDot; /* Generalized Coordinate Rates    */
                                /* number of coords                 */

```



```

/* names of coords */
/* units of coords */
/* array of pointers to coords (USER) */

RD_SIGNALSET  genCoordsDotDot; /* Generalized Coordinates Accel */
/* number of coords */
/* names of coords */
/* units of coords */
/* array of pointers to coords (USER) */

RD_SIGNALSET  genForces; /* Generalized Forces */
/* number of forces */
/* names of forces */
/* units of forces */
/* array of pointers to forces (USER) */

RD_SIGNALSET  controls; /* Controls */
/* number of controls */
/* names of controls */
/* units of controls */
/* array of pointers to controls (USER) */

int           singular; /* solution for controls singular? */

RD_SIGNALSET  inputs; /* Inputs */
/* number of inputs */
/* names of inputs */
/* units of inputs */
/* array of pointers to inputs (USER) */

RD_SIGNALSET  outputs; /* Outputs */
/* number of outputs */
/* names of outputs */
/* units of outputs */
/* array of pointers to outputs (USER) */

RD_SIGNALSET  dynOutputs; /* Outputs */
/* number of dynamic outputs */
/* names of dynamic outputs */
/* units of dynamic outputs */
/* array of pointers to dynOutputs(USER) */

} RD_DATASET;

```

```

/*****

```

```

*
*      Interface To Recursive Dynamics and Control Package
*
*
*
*
Language : ANSI C, with makeheader and personal extensions
Author   : Ross Koningstein
Date     : 14 Feb 90
Purpose  : This set of routines provides a generic interface to the
           Recursive Dynamics (RD) package for Computed Torque control
           of rigid body systems.
*/

/* RDCreateRobot
   name       : a string which is your name for this controlled system
   filename    : a string which is the filename of the file which contains
                 the configuration information.
   verbose     : !=0  -> verbose on file read
                 ==0  -> not verbose
               - This procedure creates data structures for a robot dynamical
                 model. It accepts a filename for a configuration file, and will open
                 this file, read its contents, and close it.
               - Data interface with user procedures occurs through the returned
                 dataset: values may be read out of or written into this set. The dataset
                 contains all the information used by all other 'RD' functions.

   returns    : pointer to dataset -> okay
                 NULL              -> problem with config file.
*/
extern RD_DATASET *RDCreateRobot( char *name, char *filename, int verbose );

/* RDKinematics
   RDDDataSet : dataset for controlled system.
               - kinematics terms evaluated
               - output terms (position, velocity) are valid after this
                 procedure has completed.

   returns    : !=0  -> okay
                 ==0  -> dataset 'q','qDot' pointers are NULL
*/
extern int RDKinematics( RD_DATASET *RDDDataSet );

```

```
/* RDJacobian
   RDDataset :   dataset for controlled system.
               - Jacobian equation numerically formed
   returns    :   !=0 -> okay
               ==0 -> dataset 'specs' pointer is NULL
*/
extern int RDJacobian( RD_DATASET *RDDataset );

/* RDComputedTorque
   RDDataset :   dataset for controlled system.
               - Jacobian equation evaluated
               - Jacobian equation solved for generalized accelerations
                 (ERROR) singularity in Jacobian equation
               - inverse dynamics used to compute joint torques
   returns    :   !=0 -> okay
               ==0 -> singularity (all joint torques, forces = 0 )
               ==0 -> *or* dataset 'Q' pointer is NULL
*/
extern int RDComputedTorque( RD_DATASET *RDDataset );

/* RDDynamics
   RDDataset :   dataset for simulated system.
               - Dynamics equation numerically formed
   returns    :   !=0 -> okay
               ==0 -> dataset 'Q' pointer is NULL
*/
extern int RDDynamics( RD_DATASET *RDDataset );

/* RDStateDerivative
   RDDataset :   dataset for simulated system.
               - Dynamics equation numerically solved
   returns    :   !=0 -> okay
               ==0 -> dataset 'specs' pointer is NULL
*/
extern int RDStateDerivative( RD_DATASET *RDDataset );

/* RDEnergy
   RDDataset :   dataset for system.
               - FIRST do Kinematics using state
               - AND Dynamics equation must be formulated
```

```

        - THEN this routine can be used to determine the energy in the system
returns   :   !=0  -> okay
           :   ==0  -> dataset 'specs' pointer is NULL
*/
extern double RDEnergy( RD_DATASET *RDDDataSet );

/* RDChangeMass
   RDDDataSet :   dataset for system.
   BodyName   :   name of body.
   mass       :   new mass value for body.
               - change mass of a body - allow for different dynamics
returns      :   !=0  -> okay
               :   ==0  -> dataset 'specs' pointer is NULL
*/
extern int RDChangeMass( RD_DATASET *RDDDataSet, char *bodyName, double mass );

/* RDChangeInertia
   RDDDataSet :   dataset for system.
   BodyName   :   name of body.
   mass       :   new inertia value for body.
               - change inertia of a body - allow for different dynamics
returns      :   !=0  -> okay
               :   ==0  -> dataset 'specs' pointer is NULL
*/
extern int RDChangeInertia( RD_DATASET *RDDDataSet,
                           char *bodyName, double inertia );

/* RDPrintCoords
   RDDDataSet :   dataset for controlled system.
               - prints generalized coordinate names and values to stdout.
*/
extern void RDPrintCoords( RD_DATASET *dataP );

/* RDPrintForces
   RDDDataSet :   dataset for controlled system.
               - prints generalized forces' names and values to stdout.
*/
extern void RDPrintForces( RD_DATASET *dataP );

/* RDPrintControls
   RDDDataSet :   dataset for controlled system.
               - prints control specification names and values to stdout.

```

```
*/
extern void RDPrintControls( RD_DATASET *dataP );

/* RDPrintInputs
   RDDDataSet :   dataset for controlled system.
                 - prints input signal names and values to stdout.
*/
extern void RDPrintInputs( RD_DATASET *dataP );

/* RDPrintOutputs
   RDDDataSet :   dataset for controlled system.
                 - prints output signal names and values to stdout.
*/
extern void RDPrintOutputs( RD_DATASET *dataP );

/* RDPrintDynOutputs
   RDDDataSet :   dataset for controlled system.
                 - prints dynamic output signal names and values to stdout.
*/
extern void RDPrintDynOutputs( RD_DATASET *dataP );

/* RDPrintJacobian
   RDDDataSet :   dataset for controlled system.
                 - prints Jacobian matrix equation to stdout.
*/
extern void RDPrintJacobian( RD_DATASET *dataP );

/* RDPrintDynamics
   RDDDataSet :   dataset for controlled system.
                 - prints inertia (mass) matrix equation to stdout.
*/
extern void RDPrintDynamics( RD_DATASET *dataP );

#endif /* rd_h */
```

C.2 Matlab Interface Specification

This interface specification comes from the matlab include file *rdh.m*, which should be run prior to calling any RD functions from within Matlab. It sets up all the function codes (e.g., RD_INIT) that are used to access RD routines.

```
%-----          RD function references          -----
%
%   Ross Koningstein
%   Recursive Dynamics Package for Matlab
%
%=====

% initialize a system: reads in a configuration file
% [ns,ni,no,ndo,nc] = rd( RD_INIT, modelNumber, 'filename' );
RD_INIT          = 1;

% do kinematics: uses model and state
% sets up partial velocities, computes outputs
% outputs = rd( RD_KINEMATICS, modelNumber, state );
RD_KINEMATICS    = 2;

% formulate Jacobian equation (for CT control)
% [] = rd( RD_JACOBIAN, modelNumber, controls);
RD_JACOBIAN     = 3;

% perform CT control: solve for accelerations and then torques
% [actuators,dynoutputs] = rd( RD_COMPUTED_TORQUE, modelNumber );
RD_COMPUTED_TORQUE = 4;

% setup dynamics equation ( $\dot{\mu} = -\nu + \tau$ )
% [] = rd( RD_DYNAMICS, modelNumber, actuators );
RD_DYNAMICS     = 5;

% solve simulation equations for state derivative
% [state_deriv, dynOutputs] = rd( RD_SIMULATION, modelNumber )';
RD_SIMULATION   = 6;

% evaluate energy, given dynamics equation
% energy = rd( RD_ENERGY, modelNumber );
RD_ENERGY       = 7;

% print generalized coordinates (with names)
```

```
% [] = rd( RD_PRINT_COORDS, modelNumber );
RD_PRINT_COORDS      = 8;

% print generalized forces (with names)
% [] = rd( RD_PRINT_FORCES, modelNumber );
RD_PRINT_FORCES      = 9;

% print output signals (with names)
% [] = rd( RD_PRINT_OUTPUTS, modelNumber );
RD_PRINT_OUTPUTS     = 10;

% print dynamic output signals -- accelerations (with names)
% [] = rd( RD_PRINT_DYN_OUTPUTS, modelNumber );
RD_PRINT_DYN_OUTPUTS = 11;

% print Jacobian equation  $J \dot{u} = -J_d u + A_{des}$ 
% [] = rd( RD_PRINT_JACOBIAN, modelNumber );
RD_PRINT_JACOBIAN    = 12;

% print Dynamics equation  $M \ddot{u} = -N u + F$ 
% [] = rd( RD_PRINT_DYNAMICS, modelNumber );
RD_PRINT_DYNAMICS     = 13;

% change parameters in dynamical model
% change the mass of a body
% mass = rd( RD_CHANGE_MASS, modelNumber, bodyName, mass );
RD_CHANGE_MASS        = 14;
% change the inertia of a body
% inertia = rd( RD_CHANGE_MASS, modelNumber, bodyName, inertia );
RD_CHANGE_INERTIA     = 15;
% change the location of a point on a body
% location = rd( RD_CHANGE_MASS, modelNumber, pointName, location );
% RD_CHANGE_LOCATION = 16;

% get parameters of dynamical model
% get the mass of a body
% mass = rd( RD_GET_MASS, modelNumber, bodyName );
RD_GET_MASS           = 17;
% get the inertia of a body
% inertia = rd( RD_GET_INERTIA, modelNumber, bodyName );
RD_GET_INERTIA        = 18;
% get the location of a point on a body
% location = rd( RD_GET_LOCATION, modelNumber, pointName );
RD_GET_LOCATION       = 19;
```

```
% get system matrices so you can toy around with them
% get Jacobian matrix
% Jacobian = rd( RD_GET_JACOBIAN, modelNumber );
RD_GET_JACOBIAN    = 20;
% get Mass matrix
% MassMatrix = rd( RD_GET_DYNAMICS, modelNumber );
RD_GET_DYNAMICS    = 21;
```


C.3 Evaluation of Kinematics

This ANSI C source code comes from the file *kin.c*. It covers the initialization routines, including the recursive routines that evaluate mass sums in kinematic chains for 2D systems. Kinematics evaluation is also a recursive routine. It effectively uses partial velocities from the previous link and modifies individual partials according to its articulation.

```

/*****
*
*           F O R W A R D       K I N E M A T I C S
*
*
*
*
*   Language :  C with personal macro extensions
*   Author   :  Ross Koningstein
*   Date    :  15 Jan 89
*   Purpose  :  These routines calculate all of the partial velocities
*               and their derivatives for (v,w,L,H).
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "c.ext"
#include "robot2d.h"
#include "vec_math2d.h"
#include "functions.h"

/* local function prototypes */
void  KinPrelim( ROBOT *robotP, DLIST_ELEMENT *bodyListP, integer layer );
double MassSum( DLIST_ELEMENT *bodyListP );
void  Kinematics( ROBOT *robotP, BODY *BodyP, integer layer );

void KinematicsInit( ROBOT *robotP, integer verbose )
{
    register BODY      *inertialP = robotP->bodyP;
    register integer    j;

    if ( verbose )
        printf("RD> Kinematics init  of order %d system\n", robotP->order );
    /* inertial start point and base: orientation never changes */
    inertialP->b1.x = 1.0;      inertialP->b1.y = 0.0;

```

```

inertialP->b2.x = 0.0;    inertialP->b2.y = 1.0;

vec_zero2d( &inertialP->r0) ;
vec_zero2d( &inertialP->v0) ;
robotP->massSum = 0.0;
inertialP->genSpd = NO_GENSPD;
for ( j=0; j<robotP->order; j++ )
{
    /* partial velocities */
    vec_zero2d( &inertialP->v[j]) ;
    vec_zero2d( &inertialP->vd[j]) ;
    vec_zero2d( &inertialP->vcm[j]) ;
    vec_zero2d( &inertialP->vcmd[j]) ;
    inertialP->mSum[j] = 0.0;
}

/* zero out partial momenta */
for ( j=0; j<robotP->order; j++ )
{
    robotP->H[j]      = 0.0;
    robotP->Hd[j]     = 0.0;
    robotP->L[j].x    = 0.0;
    robotP->L[j].y    = 0.0;
    robotP->Ld[j].x   = 0.0;
    robotP->Ld[j].y   = 0.0;
}

if ( verbose )
printf("\nKinematics partial velocity dependencies\n");
KinPrelim( robotP, inertialP->outboard, 0 );

if ( verbose )
printf("\nKinematics partial linear momentum dependencies\n");
MomPrelim( robotP, inertialP->outboard, 0 );

if ( verbose )
{
    printf("done\n");
    printf("-----\n");
}
}

void KinPrelim( ROBOT *robotP, DLIST_ELEMENT *bodyListP, integer layer )
{

```

```

integer    r;
integer    s;
BODY       *bodyP;

/* do until end of chain */
while ( bodyListP )
{
    bodyP = bodyListP->bodyP;

    /* update local endpoint, cm partial velocities, and derivatives */
    if ( bodyP->jointType is DISCONNECTED_JOINT )
    {
        r = bodyP->genSpd;
        for( s=0; s<robotP->order; s++ )
        {
            vec_zero2d( &bodyP->v[s] );
            vec_zero2d( &bodyP->vcm[s] );
        }
        /* setup (constant) partial velocities for base */
        vec_copy2d( &bodyP->v[r-2], &robotP->bodyP->b1 );
        vec_copy2d( &bodyP->vcm[r-2], &robotP->bodyP->b1 );
        vec_copy2d( &bodyP->v[r-1], &robotP->bodyP->b2 );
        vec_copy2d( &bodyP->vcm[r-1], &robotP->bodyP->b2 );
    }

    if ( bodyP->bodyType is LINK ) then
    {
        /* do other branches in tree */
        KinPrelim( robotP, bodyP->outboard, layer+1 );
    }
    bodyListP = bodyListP->listP;
}

}

void MomPrelim( ROBOT *robotP, DLIST_ELEMENT *bodyListP, integer layer )
{
    register    integer    s;
    register BODY *bodyP;
    register BODY *inBodyP;

    /* use partial velocity dependencies to simplify momentum calculation */
    while ( bodyListP is_not NULL ) then
    {
        bodyP = bodyListP->bodyP;

```

```

inBodyP = bodyP->inboard;
if ( bodyP->bodyType is LINK ) then
{
    /* free flying robot mass sum */
    robotP->massSum += bodyP->mass;

    if ( bodyP->jointType is DISCONNECTED_JOINT ) then
    {
        s = bodyP->genSpd;
        /* add all mass with this partial velocity in this chain */
        bodyP->mSum[s-1] = bodyP->mass + MassSum( bodyP->outboard );
        bodyP->mSum[s-2] = bodyP->mSum[s-1];

        /* setup (constant) linear momentum terms for x,y */
        vec_scale2d( &robotP->L[s-2],bodyP->mSum[s-2],&bodyP->v[s-2] );
        vec_scale2d( &robotP->L[s-1],bodyP->mSum[s-1],&bodyP->v[s-1] );
        /* Ld terms here are zero */

    }
    else
    {
        s = inBodyP->genSpd;
        if ( s is_not NO_GENSPD ) then
        {
            /* add mass with this partial velocity in this chain */
            bodyP->mSum[s] = bodyP->mass+MassSum( bodyP->outboard );
        }
    }
    MomPrelim( robotP, bodyP->outboard, layer+1 );
}

bodyListP = bodyListP->listP;
}

double MassSum( DLIST_ELEMENT *bodyListP )
{
    BODY      *bodyP;
    double     mass;

    mass = 0.0;
    /* if bodies have mass, and same partial velocity, add mass */
    while ( bodyListP is_not NULL )

```

```

        {
            bodyP = bodyListP->bodyP;
            mass += bodyP->mass + MassSum( bodyP->outboard );
            bodyListP = bodyListP->listP;
        }
    return mass;
}

void KinematicsEval( register ROBOT *robotP )
{
    register    integer    j;
    double      invMass;
    VECTOR      *Lr, *Lrd;
    double      *Hr, *Hrd;
    double      *U;
    DLIST_ELEMENT *listP;

    for ( j=2; j<robotP->order; j++ )
    {
        robotP->L[j].x = 0.0;
        robotP->L[j].y = 0.0;
        robotP->Ld[j].x = 0.0;
        robotP->Ld[j].y = 0.0;
    }
    robotP->Rcm.x = 0.0;
    robotP->Rcm.y = 0.0;

    listP = robotP->bodyP->outboard;
    while ( listP )
    {
        Kinematics( robotP, listP->bodyP, 0 );
        listP = listP->listP;
    }

    /* Calculate Momentum Terms if robot is Free-Flying */
    /* get robot's center of mass position */
    invMass = 1.0/robotP->massSum;
    vec_scale2d( &robotP->Rcm, invMass, &robotP->Rcm );

    /* evaluate linear momentum(<-FIX!), angular momentum */
    robotP->Hcm = 0.0;
    robotP->Lcm.x = 0.0;
    robotP->Lcm.y = 0.0;
    Lr = &robotP->L[2];

```

```

Hr = &robotP->H[2];
U = &robotP->u[2];
for ( j=robotP->order-2; j>0; j-- )
{
    vec_incr_sca2d( &robotP->Lcm, *U, Lr );
    robotP->Hcm += (*Hr++) * (*U);
    Lr++; U++;
}
vec_scale2d( &robotP->Vcm, invMass, &robotP->Lcm );

/* correct angular momentum to center of mass (from Inertial 0,0) */
Lr = &robotP->L[2];
Lrd = &robotP->Ld[2];
Hr = &robotP->H[2];
Hrd = &robotP->Hd[2];
for ( j=robotP->order-2; j>0; j-- )
{
    *Hr -= vec_cross2d( &robotP->Rcm, Lr );
    *Hrd -= vec_cross2d( &robotP->Vcm, Lr )
+ vec_cross2d( &robotP->Rcm, Lrd );
    Hr++; Hrd++; Lr++; Lrd++;
}

}

void Kinematics( ROBOT *robotP, register BODY *bodyP, integer layer )
{
    register      integer      r = bodyP->genSpd;
    register      BODY         *inBodyP = bodyP->inboard;
    register      integer      s = inBodyP->genSpd;
    VECTOR        pv;
    VECTOR        rm;
    double        sq,cq;
    DLIST_ELEMENT *bodyListP;
    JOINT_TYPE     jointType = bodyP->jointType;

    /* Relative and Absolute position and velocity of startpoint */
    vec_coord2d( &bodyP->r, &bodyP->start, &inBodyP->b1, &inBodyP->b2 );
    vec_add2d( &bodyP->r0, &inBodyP->r0, &bodyP->r );
    vec_copy2d( &bodyP->r0Calc, &bodyP->r0 );

    if ( bodyP->r0IsMeasured ) then
    {
        vec_copy2d( &bodyP->r0, &bodyP->r0Measured );
    }

```

```

    }

    if ( jointType is DISCONNECTED_JOINT )
    {
        vec_incr_sca2d( &bodyP->r0, robotP->q[r-2], &inBodyP->b1 );
        vec_incr_sca2d( &bodyP->r0, robotP->q[r-1], &inBodyP->b2 );
        vec_scale2d( &bodyP->v0, robotP->u[r-2], &inBodyP->b1 );
        vec_incr_sca2d( &bodyP->v0, robotP->u[r-1], &inBodyP->b2 );
    }
    else
    {
        integer      nBytes = robotP->order * sizeof(VECTOR);

        vec_copy2d( &bodyP->v0, &inBodyP->v0 );
        vec_partial2d( &pv, &bodyP->r );
        vec_incr_sca2d( &bodyP->v0, robotP->u[s], &pv );

        /* Partial Velocities, and PV Derivatives of startpoint */
        memcpy( &bodyP->v, &inBodyP->v, nBytes );
        memcpy( &bodyP->vd, &inBodyP->vd, nBytes );
        memcpy( &bodyP->vcm, &inBodyP->vcm, nBytes );
        memcpy( &bodyP->vcmd, &inBodyP->vcmd, nBytes );

        /* revolute joint - start partial velocity (create local vr) */
        vec_copy2d( &bodyP->v[s], &pv );

        /* rotary joint - start partial velocity derivative */
        vec_deriv2d( &bodyP->vd[s], &bodyP->v[s], robotP->u[s] );
    }

    /* base - calculate local [b1, b2] based on rotation and translation */
    if ( bodyP->bodyType is LINK )
    {
        if ( bodyP->object ) then
        {
            BODY      *constraintP;
            VECTOR     dr,dv;
            double     magRSq, magR;
            VECTOR     pvc;
            /* estimate body orientation and rate */
            constraintP = bodyP->constraintPoint;

            /* 1) get difference in endpoint positions */
            vec_copy2d( &dr, &constraintP->constraintBody->r0 );

```

```

vec_decr2d( &dr, &bodyP->r0 );

/* 2) normalize this distance */
magRSq = vec_dot2d( &dr, &dr );
magR = sqrt( magRSq );
vec_scale2d( &dr, 1.0/magR, &dr );
vec_copy2d( &constraintP->rho, &dr );
constraintP->length = magR;
/* 3) rotate this vector by -phi to get b1 */
bodyP->b1.x = constraintP->cPhi * dr.x
            - constraintP->sPhi * dr.y;
bodyP->b1.y = constraintP->sPhi * dr.x
            + constraintP->cPhi * dr.y;
/* 4) determine b2 from b1 */
bodyP->b2.x = -bodyP->b1.y;
bodyP->b2.y = bodyP->b1.x;

/* 5) rotate b1 by inboard (-)b1,b2 to get cq,sq */
cq = inBodyP->b1.x * bodyP->b1.x + inBodyP->b1.y * bodyP->b1.y;
sq = inBodyP->b2.x * bodyP->b1.x + inBodyP->b2.y * bodyP->b1.y;
/* 6) determine q using arctan 4 quadrant */
robotP->q[r] = atan2( sq, cq );
bodyP->angle = robotP->q[r];
/* 7) get difference in endpoint velocities */
vec_copy2d( &dv, &constraintP->constraintBody->v0 );
vec_decr2d( &dv, &bodyP->v0 );
/* 8) use partial velocity to determine rate */
vec_deriv2d( &pvc, &constraintP->rho, magR );
robotP->u[r] = vec_dot2d( &dv, &pvc ) / magRSq;
robotP->qDot[r] = robotP->u[r] - robotP->u[inBodyP->genSpd];
bodyP->angleRate = robotP->qDot[r];
}
else
{
    bodyP->angle = robotP->q[r];
    bodyP->angleRate = robotP->qDot[r];
    if ( jointType is DISCONNECTED_JOINT) then
    {
        robotP->u[r] = bodyP->angleRate;
        robotP->u[r-1] = robotP->qDot[r-1];
        robotP->u[r-2] = robotP->qDot[r-2];
    }
    else
    {

```



```

        robotP->u[r]    = bodyP->angleRate + robotP->u[s];
    }
    /* rotate inboard body vectors by angle */
    sq = sin( bodyP->angle );
    cq = cos( bodyP->angle );

    /* calculate b1 vector */
    bodyP->b1.x = cq * inBodyP->b1.x + sq * inBodyP->b2.x;
    bodyP->b1.y = cq * inBodyP->b1.y + sq * inBodyP->b2.y;

    /* calculate b2 vector */
    bodyP->b2.x = - bodyP->b1.y;
    bodyP->b2.y =  bodyP->b1.x;
}
}

/* Relative and Absolute position, Partial Velocities,
   and Partial Velocity Derivatives of center of mass */
if ( bodyP->bodyType is LINK ) then
{
    vec_coord2d( &bodyP->rcm, &bodyP->cm, &bodyP->b1, &bodyP->b2 );
    vec_add2d( &bodyP->r0cm, &bodyP->r0, &bodyP->rcm );

    /* center of mass sum */
    vec_incr_sca2d( &robotP->Rcm, bodyP->mass, &bodyP->r0cm );

    /* rotary joint - cm partial velocity */
    vec_partial2d( &bodyP->vcm[r], &bodyP->rcm );
    vec_incr_sca2d( &robotP->L[r], bodyP->mass, &bodyP->vcm[r] );
    /* rotary joint - cm derivative of partial velocity */
    vec_deriv2d( &bodyP->vcmd[r], &bodyP->vcm[r], robotP->u[r] );
    vec_incr_sca2d( &robotP->Ld[r], bodyP->mass, &bodyP->vcmd[r] );

    /* linear momentum due to stuff along this chain */
    if ( jointType is_not DISCONNECTED_JOINT ) then
    {
        /* cm partials */
        vec_copy2d( &bodyP->vcm[s], &bodyP->v[s] );
        vec_copy2d( &bodyP->vcmd[s], &bodyP->vd[s] );

        /* partial velocity 's' affects all masses on this chain */
        vec_incr_sca2d( &robotP->L[s], bodyP->mSum[s], &bodyP->v[s] );
        /* Ld terms here are non-zero */
    }
}

```

```

        vec_incr_sca2d( &robotP->Ld[s], bodyP->mSum[s], &bodyP->vd[s] );
    }

    vec_scale2d( &rm, bodyP->mass, &bodyP->r0cm );
    robotP->H[r] = bodyP->inertia + vec_cross2d( &rm, &bodyP->vcm[r] );
    robotP->Hd[r] = 0.0;

    while ( (s=inBodyP->genSpd) is_not NO_GENSPD )
    {
        /* add angular momenta components */
        robotP->H[s] += vec_cross2d( &rm, &bodyP->vcm[s] );
        robotP->Hd[s] += vec_cross2d( &rm, &bodyP->vcmd[s] );
        inBodyP = inBodyP->inboard;
    }
}

/* do constraint points for objects */
if ( bodyP->object ) then
{
    Kinematics( robotP, bodyP->constraintPoint, layer+1 );
}

/* do until end of outboard chain(s) */
bodyListP = bodyP->outboard;
while ( bodyListP is_not NULL ) then
{
    Kinematics( robotP, bodyListP->bodyP, layer+1 );
    bodyListP = bodyListP->listP;
}
}

```

C.4 Evaluation of Dynamical Equations of Motion

This C source code comes from the file *dyn.c*.

```

/*****
*
*           F O R W A R D       D Y N A M I C S
*
*
*
Language : C with personal macro extensions
Author   : Ross Koningstein
Date     : 13 Apr 89

```

Purpose : These routines evaluate the dynamical equations of motion.
 */

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include "c.ext"
#include "robot2d.h"
#include "vec_math2d.h"
#include "functions.h"
```

/* local function prototypes */

```
void Dynamics( ROBOT *robotP, BODY *BodyP, integer layer );
void Diagonal( ROBOT *robotP, BODY *BodyP, integer layer );
void EndpointConstraint(
    CONSTRAINT *constraint,
    double      *rowX,
    double      *rowY,
    double      *u,
    integer      order,
    integer      order_c );
```

```
void DynamicsInit( ROBOT *robotP, integer verbose )
```

```
{
    BODY      *inertialP = robotP->bodyP;
    DLIST_ELEMENT *listP;

    if ( verbose )
        printf("RD> Simulation init of order %d system\n", robotP->order );

    /* Calculate the diagonal of the Inertia (mass) matrix */
    listP = inertialP->outboard;
    while ( listP )
    {
        Diagonal( robotP, listP->bodyP, 0 );
        listP = listP->listP;
    }
}
```

```
void Diagonal( ROBOT *robotP, BODY *bodyP, integer layer )
```

```
{
    register      BODY      *outBodyP;
```

```

register      integer      r = bodyP->genSpd;
DLIST_ELEMENT *bodyListP;

if ( bodyP->bodyType is LINK ) then
{
    if ( bodyP->jointType is DISCONNECTED_JOINT ) then
    {
        /* start of free-flying chain - elements are sum of masses */
        robotP->MDiag[r-2] = bodyP->mSum[r-2];
        robotP->MDiag[r-1] = bodyP->mSum[r-1];
    }

    /* contribution of I, mass*cm^2 of this body */
    robotP->MDiag[r] = bodyP->inertia0;

    /* contributions of outward chains */
    bodyListP = bodyP->outboard;
    while ( bodyListP is_not NULL ) then
    {
        outBodyP = bodyListP->bodyP;
        robotP->MDiag[r] += outBodyP->mSum[r]
                        * vec_dot2d( &outBodyP->start, &outBodyP->start );
        bodyListP = bodyListP->listP;
    }
}

/* do until end of chain */
bodyListP = bodyP->outboard;
while ( bodyListP is_not NULL ) then
{
    Diagonal( robotP, bodyListP->bodyP, layer+1 );
    bodyListP = bodyListP->listP;
}

}

void DynamicsEval( RD_DATASET *dataP )
{
    ROBOT      *robotP = dataP->robotP;
    register integer r,s;
    DLIST_ELEMENT *listP;
    CONSTRAINT *constraints = robotP->constraints;
    double      *u = robotP->u;
    integer      order = robotP->order + dataP->c;
    double      *nonLinearLX      = &robotP->M[0][order];

```

```

double  *nonLinearLY      = &robotP->M[1][order];
integer  row;

/* Zero out Mass (Inertia) Matrix and NL vector terms, set diagonal */
for ( r=order-1; r>=0; r-- )
{
    robotP->Mrows[r] = &robotP->M[r][0];
    for ( s=order; s>=0; s-- )
    {
        robotP->M[r][s] = 0.0;
    }
    if ( r <= robotP->order ) then
    {
        robotP->M[r][r] = robotP->MDiag[r];
    }
}

if ( robotP->robotType is ROBOT_FREE ) then
{
    /* free-flying robot - first two rows are linear momentum */
    for ( r=0; r<robotP->order; r++ )
    {
        /* enter partial linear momenta along x and y rows in Jacobian */
        robotP->M[0][r] = robotP->L[r].x;
        robotP->M[r][0] = robotP->M[0][r];
        robotP->M[1][r] = robotP->L[r].y;
        robotP->M[r][1] = robotP->M[1][r];
        /* (minus) sum derivatives for nonlinear linear momentum terms */
        *nonLinearLX -= robotP->Ld[r].x * u[r];
        *nonLinearLY -= robotP->Ld[r].y * u[r];
    }
    /* add base force terms */
    *nonLinearLX -= robotP->Q[0];
    *nonLinearLY -= robotP->Q[1];
}

/* Calculate terms in matrix */
listP = robotP->bodyP->outboard;
while ( listP )
{
    Dynamics( robotP, listP->bodyP, 0 );
    listP = listP->listP;
}

```

```

row = robotP->order;
/* append any motion constraints */
while ( constraints )
{
    /* foreach constraint, add constraint velocity terms */
    EndpointConstraint(
        constraints,
        robotP->Mrows[row],
        robotP->Mrows[row+1],
        robotP->u,
        robotP->order,
        robotP->order+dataP->c );
    constraints = constraints->nextP;
    row += 2;
}

for ( r=robotP->order; r<order; r++ )
{
    for ( s=0; s<order; s++ )
    {
        robotP->M[s][r] = robotP->M[r][s];
    }
}

}

void EndpointConstraint(
    CONSTRAINT *constraint,
    double      *rowX,
    double      *rowY,
    double      *u,
    integer      order,
    integer      order_c )
{
    register integer i;
    BODY      *endpoint1 = constraint->body1;
    BODY      *endpoint2 = constraint->body2;
    register VECTOR *partial = endpoint1->v;
    register VECTOR *partialD = endpoint1->vd;
    double *nonLinearX      = rowX + order_c;
    double *nonLinearY      = rowY + order_c;
    double *oldRowX          = rowX;
    double *oldRowY          = rowY;
    VECTOR force; /* relaxation force at constraint */
    double Kp = constraint->Kp; /* relaxation: spring */

```

```

double    Kv = constraint->Kv; /* relaxation: damping */

*nonLinearX = 0.0;
*nonLinearY = 0.0;
for ( i=order; i>0; i-- )
{
    /* enter partial velocities along x and y rows in Jacobian */
    *rowX = partial->x;
    *rowY = partial->y;
    partial++;      rowX++;      rowY++;
    /* sum derivatives for nonlinear components */
    *nonLinearX -= partialD->x * (*u);
    *nonLinearY -= partialD->y * (*u);
    partialD++;
    u++;
}
vec_scale_add2d( &force, Kp, &endpoint1->r0, Kv, &endpoint1->v0 );

partial = endpoint2->v;
partialD = endpoint2->vd;
rowX     = oldRowX;
rowY     = oldRowY;
for ( i=order; i>0; i-- )
{
    /* enter partial velocities along x and y rows in Jacobian */
    *rowX -= partial->x;
    *rowY -= partial->y;
    partial++;      rowX++;      rowY++;
    /* sum derivatives for nonlinear components */
    *nonLinearX += partialD->x * (*u);
    *nonLinearY += partialD->y * (*u);
    partialD++;
    u++;
}
vec_incr_sca2d( &force, -Kp, &endpoint2->r0 );
vec_incr_sca2d( &force, -Kv, &endpoint2->v0 );
*nonLinearX -= force.x;
*nonLinearY -= force.y;
}

void Dynamics( ROBOT *robotP, register BODY *bodyP, integer layer )
{
    register    integer    r = bodyP->genSpd;
    register    BODY       *inBodyP = bodyP->inboard;

```

```

register      integer      s = inBodyP->genSpd;
register      integer      t;
DLIST_ELEMENT *bodyListP;

if ( bodyP->bodyType is LINK ) then
{
    /* generalized force computation */
    *robotP->Nu[bodyP->genSpd] += robotP->Q[bodyP->genSpd];
    if ( inBodyP->genSpd is_not NO_GENSPD )
    {
        *robotP->Nu[inBodyP->genSpd] -= robotP->Q[bodyP->genSpd];
    }

    /* evaluate terms involving vcm[r] - unique to this body */
    while ( (t=inBodyP->genSpd) is_not NO_GENSPD )
    {
        robotP->M[t][r] += bodyP->mass
            * vec_dot2d( &bodyP->vcm[t], &bodyP->vcm[r] );
        robotP->M[r][t] = robotP->M[t][r];

        /* nonlinear terms */
        *robotP->Nu[r] -= bodyP->mass
            *vec_dot2d( &bodyP->vcm[r], &bodyP->vcmd[t] ) * robotP->u[t];
        *robotP->Nu[t] -= bodyP->mass
            *vec_dot2d( &bodyP->vcm[t], &bodyP->vcmd[r] ) * robotP->u[r];

        /* evaluate terms involving v[s] - not unique to this body */
        if ( t is_not s ) then
        {
            robotP->M[t][s] += bodyP->mSum[s]
                * vec_dot2d( &bodyP->v[t], &bodyP->v[s] );
            robotP->M[s][t] = robotP->M[t][s];

            /* non-linear terms */
            *robotP->Nu[s] -= bodyP->mSum[s]
                *vec_dot2d( &bodyP->v[s], &bodyP->vd[t] ) * robotP->u[t];
            *robotP->Nu[t] -= bodyP->mSum[s]
                *vec_dot2d( &bodyP->v[t], &bodyP->vd[s] ) * robotP->u[s];
        }

        inBodyP = inBodyP->inboard;
    }
}

```



```
/* do until end of chain */
bodyListP = bodyP->outboard;
while ( bodyListP is_not NULL ) then
{
    Dynamics( robotP, bodyListP->bodyP, layer+1 );
    bodyListP = bodyListP->listP;
}
}
```

2

Appendix D

Multibody Simulation under Matlab

This appendix contains computer code for the RD matlab script files that perform simulations. The following Matlab scripts are included:

- The independent endpoint controller simulation
- The cooperating arm endpoint controller simulation

D.1 Independent Endpoint Control Simulation

This simulation uses the configuration files described in section 6.1 in concert with **RD** simulation code in Matlab. This Matlab file, *sim_2arm_circle.m*, reads in the description file, sets an initial state, and uses a trajectory with a CT controller to determine motor torques. These motor torques are fed in to a simulation.

```
% simulation test case for rd
clear
clg

% get function references for rd
rdh

% Dynamics and control system RD models
DYN = 1;
CON = 2;
filename = 'config.endpt';

% read in configuration files for dynamics (1) and controller (2)
[ns_d] = rd( RD_INIT, DYN, filename );
[ns_c] = rd( RD_INIT, CON, filename );

% setup state
state = [ 0.6 0 3.14  -.78 1.57  .84 -1.58    0.0 0.0 0.0  0.0 0.0  0.0 0.0];

Pdes = [ 0 0 0 0 ]';
Vdes = [ 0 0 0 0 ]';
Ades = [ 0 0 0 0 ]';
Kp = 100;
Kv = 18;

% setup controls
controls = [0 0 0 0 0 0]';

% setup actuators
actuators = [0 0 0 0 0 0]';

% Timing information
dt = 1/60;
T = 3.0;
loops = T / dt;
t = 0;
```

```
Energy = zeros(loops+1);
State = zeros(loops+1,ns_d);
time=[(0:loops)*dt];
nsteps = 6;

% plot work area
% plot_tbl
axis([-0.5,1.0,-.6,.6]);
hold on

plot([0 0],[-.7 1])
plot([-.8 .9],[0 0])
plot([-.2 .2],[.2 .2])
plot([-.2 .2],[-.2 -.2])

% loop to evaluate controller:dynamics simulation
for ( i= 1:(loops+1) )
    % save state
    State(i,:) = state;

    % plot vehicle state over simulation run
    if ( rem( i-1, nsteps ) == 0 )
        plotacv( state );
    end

    % trajectory generation
    [Pdes,Vdes,Ades] = traj(t);

    % do kinematics
    outputs = rd( RD_KINEMATICS, CON, state );
    outputs = rd( RD_KINEMATICS, DYN, state );

    % evaluate desired controls
    a_des = Kp*( Pdes - outputs(1:4) ) + Kv*( Vdes - outputs(5:8) ) + Ades;

    controls = [ 0; 0; 0; a_des ]';

    % do Jacobian equation
    rd( RD_JACOBIAN, CON, controls);

    % solve computed torque
    [actuators,outputs] = rd( RD_COMPUTED_TORQUE, CON );

    % formulate dynamics equation
```

```
rd( RD_DYNAMICS, CON, actuators );
rd( RD_DYNAMICS, DYN, actuators );

% integrate state
[t,state] = ie( 'wrap', dt, t, state, actuators );
end

text(0.9,.55,'X','sc')
text(0.45,.85,'Y','sc')

hold off

'plot done'
```

D.2 Cooperative-Arm Object Control Simulation

This simulation uses the configuration files described in section 6.2 in concert with **RD** simulation code in Matlab. This Matlab file, *sim_obj.circle.m*, reads in the description file, sets an initial state, and uses a trajectory with a CT controller to determine motor torques. These motor torques are fed in to a simulation. Default values for relaxation constraints, $K_p = 625$ and $K_v = 50$, are used.

```
% simulation test case for rd
clg

% get function references for rd
rdh

% Dynamics and control system RD models
DYN = 1;
CON = 2;

dyn_file='config.obj';
con_file='config.obj';
ObjLen = 0.5;
ObjR    = 0.1;

% read in configuration files for dynamics (1) and controller (2)
[ns_d,ni_d,no_d,nc_d] = rd( RD_INIT, DYN, dyn_file );
[ns_c,ni_c,no_c,nc_c] = rd( RD_INIT, CON, con_file );

% setup state
state = [ 0.5 0 3.14  -1 2  1.11 -2 -.67   0.0 0.0 0.0  0.0 0.0  0.0 0.0 0.0];
state = [ 0.6 0 3.14  -1.002 1.706 1.002 -1.706 -.901  00 0 0  0 0 0 0 0];

Pdes = [ 0 0 1.57 ]';
Vdes = [ 0 0 0 ]';
Ades = [ 0 0 0 ]';
Kp = 49;
Kp = 100;
Kv = 14;
Kv = 22;

% setup controls
controls = [ 0 0 0 0 0 0 0 0 ]';

% setup actuators
```

```

actuators = [ 0 0 0 0 0 0 0 0 ]';

T=3;
% Timing information
dt = 1/60;
loops = T / dt;
t = 0;
%Energy = zeros(loops+1);
State = zeros(loops+1,ns_d);
Outputs = zeros(loops+1,no_d);
time=[(0:loops)*dt];
nsteps = 6;

% plot work area
% plot_tbl
axis([-0.5,1.0,-.6,.6]);
hold on

plot([0 0],[-.7 1])
plot([-.8 .9],[0 0])
plot([-.2 .2],[.2 .2])
plot([-.2 .2],[-.2 -.2])

% Circular trajectory
p0 = [ 0 0 ]';
r = 0.03;
w = 50 / 60 * 2 * 3.14;

% loop to evaluate controller:dynamics simulation
for ( i= 1:(loops+1) )
    % save state
    State(i,:) = state;

    % trajectory generation
    [PCdes,VCdes,ACdes] = traj2(p0,r,w,t);
    Pdes = [PCdes; pi/2];
    Vdes = [VCdes; 0];
    Ades = [ACdes; 0];

    % do kinematics
    outputs = rd( RD_KINEMATICS, CON, state );
    outputs = rd( RD_KINEMATICS, DYN, state );
    Outputs(i,:) = outputs';

```



```
% plot vehicle state over simulation run
if ( rem( i-1, nsteps ) == 0 )
    if ( i > 30 )
        plotacv( state );
        plotobj( outputs(1:2), outputs(3), ObjLen, ObjR );
    end
end

% evaluate desired controls
a_des = Kp*(Pdes(1:3)-outputs(1:3)) + Kv*(Vdes(1:3)-outputs(4:6)) + Ades;
controls = [ 0; 0; 0; 0; a_des; 0 ]';

% do Jacobian equation
rd( RD_JACOBIAN, CON, controls);

% solve computed torque
[actuators,dynoutputs] = rd( RD_COMPUTED_TORQUE, CON );
% do not allow thruster activity
actuators = [ 0; 0; 0; actuators(4:8) ];

% formulate dynamics equation
rd( RD_DYNAMICS, DYN, actuators );

% integrate state
[t,state] = ie( 'wrap', dt, t, state, actuators );
end

text(0.9,.55,'X','sc')
text(0.45,.85,'Y','sc')

hold off

'plot done'
```

2

Bibliography

- [1] Harold L. Alexander. *Experiments in Control of Satellite Manipulators*. PhD thesis, Stanford University, Department of Electrical Engineering, Stanford, CA 94305, December 1987.
- [2] Robert H. Cannon, Jr., Marc Ullman, Ross Koningstein, Stan Schneider, Warren Jasper, and Roberto Zanutta. NASA Semi-Annual Report on Control of Free-Flying Space Robot Manipulator Systems. Semi-Annual Report 5, Stanford University Aerospace Robotics Laboratory, Stanford, CA 94305, August 1987.
- [3] Craig R. Carignan. *Control Strategies for Manipulating Payloads in Weightlessness with a Free-Flying Robot*. PhD thesis, Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, MIT, MA, September 1987.
- [4] Craig R. Carignan and David L. Akin. Cooperative control of two arms in the transport of an inertial load in zero gravity. *IEEE Journal of Robotics and Automation*, 4(4):414-419, August 1988.
- [5] Vincent Chen. *Experiments in Adaptive Control of a Free-Flying Robot with Payload*. PhD thesis, Stanford University, Department of Aeronautics and Astronautics, Stanford, CA 94305, June 1991. To be Published.
- [6] John J. Craig. *Introduction to Robotics Mechanics and Control*. Addison-Wesley, Reading, MA, 1986.
- [7] S. Dubowski and Z. Vafa. A virtual manipulator model for space robotic systems. In *Proceedings of the Workshop on Space Telerobotics*, Pasadena, CA, July 1987.

- [8] S.A. Hayati. Dynamics and control of multipl robotic manipulators. In *Proceedings of Workshop on Space Telerobotics*, Pasadena, CA, July 1987.
- [9] Michael G. Hollars and Robert H. Cannon, Jr. Experimental implementation of a nonlinear estimator in the control of flexible joint manipulators. In *Proceedings of the IFAC Aerospace Conference*, Tsukuba, Japan, July 1989.
- [10] John M. Hollerbach. A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10(11):730-736, November 1980.
- [11] Warren J. Jasper. *Experiments in Thrusterless Robotic Locomotion Control for Space Applications*. PhD thesis, Stanford University, Department of Aeronautics and Astronautics, Stanford, CA 94305, September 1990. To be Published.
- [12] W.J. Jasper and Jr. R.H. Cannon. Initial experiments in thrusterless locomotion control of a free-flying robot. In *Proceedings of the ASME Winter Annual Meeting*, Dallas, TX, November 1990.
- [13] Thomas R. Kane and David A. Levinson. *Dynamics: Theory and Application*. McGraw-Hill Series in Mechanical Engineering. McGraw-Hill, New York, NY, 1985.
- [14] B. W. Kernigan and R. Pike. *The UNIX Programming Environment*. Prentice-Hall, 1984.
- [15] Oussama Khatib. Dynamic control of manipulators in operational space. In *Proceedings of the Sixth CISM-IFTOMM Congress on Theory of Machines and Mechanisms*, pages 1128-1131, New Delhi, India, December 1983.
- [16] Pradeep K. Khosla and Takeo Kanade. Experimental evaluation of the feedforward compensation and computed-torque control schemes. In *Proceedings of the American Control Conference*, pages 790-798, Seattle, WA, June 1986.
- [17] Ross Koningstein, Marc Ullman, and Robert H. Cannon, Jr. Computed torque control of a free-flying cooperating-arm robot. In *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, February 1989.

- [18] J. Y. S. Luh, M. W. Walker, and R. P. C. Paul. Resolved-acceleration control of mechanical manipulators. *IEEE Transactions on Automatic Control*, AC-25(3):468–474, June 1980.
- [19] J. Y. S. Luh and Yuan-Fang Zheng. Computation of input generalized forces for robots with closed-kinematic chain mechanisms. *IEEE Journal of Robotics and Automation*, RA-1(2):95–103, June 1985.
- [20] Yoshihiko Nakamura and Modjtaba Ghodoussi. A computational scheme of closed link robot dynamics derived by d'Alembert principle. In *Proceedings of the International Conference on Robotics and Automation*, pages 1354–1360, Philadelphia, PA, April 1988. IEEE, IEEE Computer Society.
- [21] Paul Eric Nielan. *Efficient Computer Simulation of Motions of Multibody Systems*. PhD thesis, Stanford University, Department of Mechanical Engineering, Stanford, CA 94305, September 1986.
- [22] Evangelos Papadopolous and Steven Dubowski. On the nature of control algorithms for space manipulators. In *Proceedings of the IEEE Robotics and Automation*, Cincinnati, OH, May 1990.
- [23] Lawrence Pfeffer, Oussama Khatib, and John Hake. Joint torque sensory feedback in the control of a PUMA manipulator. In *Proceedings of the American Control Conference*, pages 818–824, Seattle, WA, June 1986.
- [24] F.H. Rehsteiner. *Static and Dynamic Properties of Hydrostatic Thrust Gas Bearings with Curved Surfaces*. PhD thesis, Stanford University, Department of Aeronautics and Astronautics, Stanford, CA 94305, 1990.
- [25] G. Rodriguez and K. Kreutz. Recursive mass matrix factorization and inversion. Technical report, NASA Jet Propulsion Laboratory, Pasadena, CA, March 1988. JPL Publication 88–11.
- [26] Dan E. Rosenthal. Order N Formulation for Equations of Motion of Multibody Systems. In G. Man and R. Laskin, editors, *Proceedings of the Workshop on Multibody*

- Simulation*, pages 1122–1150, Pasadena, CA, April 1988. NASA Jet Propulsion Laboratory. JPL D-5190, Volume III.
- [27] Dan E. Rosenthal and Mike A. Sherman. Symbolic multibody equations via kane's method. In *Proceedings of the AAS/AIAA Astrodynamics Specialist Conference*, Lake Placid, NY, August 1983. Paper No. 83-303.
- [28] Schechter and Levinson. *AutoLev User's Guide*. Ford Aerospace, Inc., Palo Alto, CA, August 1988. Internal Product.
- [29] S. Schneider. *Experiments in the Dynamic and Strategic Control of Cooperating Manipulators*. PhD thesis, Stanford University, Stanford, CA 94305, July 1989.
- [30] H. Seraji. Adaptive hybrid control of manipulators. In *Proceedings of Workshop on Space Telerobotics*, Pasadena, CA, July 1987.
- [31] S. W. Tilley, M. G. Hollars, and K. S. Emerick. Experimental control results in a compact space robot actuator. In *Proceedings of the ASME Winter Annual Meeting*, San Francisco, CA, December 1989.
- [32] X. Yun T.J. Tarn and A.K. Bejczy. Nonlinear feedback control of multiple robot arms. In *Proceedings of Workshop on Space Telerobotics*, Pasadena, CA, July 1987.
- [33] Christopher R. Uhlik. *Experiments in High-Performance Nonlinear and Adaptive Control of a Two-Link, Flexible-Drive-Train Manipulator*. PhD thesis, Stanford University, Department of Electrical Engineering, Stanford, CA 94305, May 1990.
- [34] M.A. Ullman and Jr. R.H. Cannon. Experiments in global navigation and control of a free-flying space robot. In *Proceedings of the ASME Winter Annual Meeting, Dynamics and Control of Multibody Robotic Systems with Space Applications*, pages 37–44, San Francisco, CA, December 1989.
- [35] Marc Ullman. *Experiments in Global Navigation and Control of a Free-Flying Robot*. PhD thesis, Stanford University, Department of Aeronautics and Astronautics, Stanford, CA 94305, June 1991. To Be Published.

- [36] Y. Umetani and K. Yoshida. Continuous path control of space manipulators mounted on omv. *ACTA Astronautica*, 15(12):981-986, 87.
- [37] Yoji Umetani and Kazuya Yoshida. Experimental study on two-dimensional free-flying robot satellite model. In *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, January 1989.
- [38] P.Th.L.M. van Woerkom and M. Guelman. Dynamics modelling, Simulation, and Control of a Spacecraft/Manipulator System. Technical Report 1, National Aerospace Laboratory, The Netherlands, Amsterdam, NL, August 1987.
- [39] Charles W. Wampler. *Computer Methods in Manipulator Kinematics, Dynamics, and Control: A Comparative Study*. PhD thesis, Stanford University, Department of Mechanical Engineering, Stanford, CA 94305, December 1984.
- [40] K. Nagai Y. Nakamura and T. Yoshikawa. Mechanics of coordinative manipulation by multiple robotic mechanisms. In *IEEE Conference on Robotics and Automation*, Raleigh, NC, March 1987.
- [41] Fumio Miyazaki Yasuhiro Masutani and Suguru Arimoto. Modeling and sensory feedback control for space manipulators. In *Proceedings of the NASA Conference on Space Telerobotics*, Pasadena, CA, January 1989.

